ترجمه إلى العربية:

عبد اللطيف محمد أديب ايمش

سطر أوامر لينكس

سطر أوامر لينُكس

The Linux Command Line

لمؤلفه:

Willam E. Shotts, Jr.

ترجمه إلى العربية:

عبد اللطيف محمد أديب ايمش

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ


Creative Commons

المحتويات باختصار

جدول المحتويات

الملحق أ: مصادر إضافية....................515

المقدمة

الحمد لله رب العالمين، وأفضل الصلاة وأتمُ التسليم على سيدنا محمدٍ إمام المرسلين وخاتم النبيين وعلى آله وصحبه أجمعين.

لا يخفى على أحدٍ التطور الكبير الذي شهده نظام لينُكس في الآونة الأخيرة، حيث ازداد انتشاره ازديادًا كبيرًا بين مستخدمي الأجهزة المكتبية، بعد أن حقق نجاحًا باهرًا في سوق الخوادم والشبكات. لكن الوافدين الجُدد على هذا النظام يصدمون بحاجز قلّة المصادر العربية التي تتحدث عنه، شارحةً ميزاته، ومُبيّنةً مرونته وتفوقه على نُظراءه. فجاء هذا الكتّاب سادًّا تلك الفجوة، مُمسكًا بيد المبتدئ في سطر الأوامر، موصلًا إياه إلى الاحتراف. يتدرج شرحه من الأساسيات إلى المواضيع المتقدمة.

آمل أن يكون هذا الكتّاب إضافةً مُفيدةً للمكتبة العربية، وأن يُفيد القارئ العربي في تعلم أحد أشهر وأقوى أنظمة التشغيل في العالم. والله وليُّ التوفيق.

عبد اللطيف محمد أديب ايمش

2014\7\23

حلب

بدعم من:



تُرِكَتْ هَذِهِ الصَفْحَةُ فَارِغَةً عَمْدًا

تمهيد

أريد إخبارك قصةً... لا، ليست قصة "كيف": في 1991، كتب لينوس تورفالدس الإصدار الأول من نواة لينُكس. يمكنك قراءة تلك القصة في كتب عديدة تتحدث عن لينُكس. ولست أنوي الحديث عن قصة "غنو". منذ عدة سنوات، بدأ ريتشارد ستالمان مشروع غنو (GNU) لإنشاء نظام حر شبيه بِيونكس (Unix-like)، صحيح أن هذه القصة مهمة أيضًا، لكن أغلب كتب لينُكس الأُخرى تحويها.

أريد إخبارك قصةً عن كيفية استعادة السيطرة على حاسوبك.

عندما بدأت مشواري مع الحواسيب عندما كنت طالبًا في أواخر 1970؛ كانت تجري في ذاك الوقت ثورة في عالم الحواسيب والتقنية: اختراع المعالج الصغري. سمح هذا النوع من المعالجات للأشخاص العاديين مثلك ومثلي باقتناء حاسوب خاص بهم. يصعب للعديد من الأشخاص في الوقت الراهن تخيل كيف كان العالم عندما كانت الشركات الكبيرة والمؤسسات الحكومية هي فقط من تستطيع تشغيل الحواسيب.

اختلف الوضع تمامًا في هذه الأيام. الحواسيب في كل مكان، بدءًا من الحواسيب المصغرة الموجودة في ساعات اليد وانتهاءً بالمراكز الضخمة للمعلومات وبالطبع كل ما بينهما. وبالإضافة إلى انتشار الحواسيب؛ فقد انتشرت معها الشبكات التي تربط ما بين هذه الحواسيب. وهذا ما فتح المجال أمام عصر جديد من عصور التقنية. ولكن كانت هنالك شركة وحيدة كبيرة في العقدين الماضيين تحاول فرض سيطرتها على حواسيب العالم وتقرر ما الذي تستطيع وما الذي لا تستطيع فعله مع الحواسيب. ولحسن الحظ، هنالك أشخاص من مختلف أنحاء العالم يقومون بأشياء كثيرة للتصدي لها. إنهم يحاربون ليبقوا مسيطرين على حواسيبهم وذلك بكتابة برمجياتهم الخاصة. إنهم يبنون لينُكس!

يتحدث أشخاصٌ عديدون عن "الحرية" عندما يشيرون إلى لينُكس، لا أظن أن أغلب الأشخاص يعلمون ما هو المعنى الحقيقي للحرية. الحرية هي المقدرة على التحكم بما يفعله حاسوبك. الحرية هي أن يكون حاسوبك بدون أيّة أسرار وتستطيع معرفة أي شيء فيه لطالما أنك مكترث لذلك.

لماذا أستخدم سطر الأوامر؟

هل لاحظت من قبل "المخترق الخارق" (كما تعلم، الشخص الذي يستطيع اختراق أي حاسوب مؤمن تأمينًا كبيرًا تابعًا لإحدى الجهات العسكرية خلال ثلاثين ثانية) في الأفلام الذي يعمل على حاسوبه دون أن يلمس الفأرة! ذلك لأن صانعي الأفلام يدركون أننا بديهيًّا نعتبر أن الطريقة الوحيدة لإنجاز الأعمال على الحاسوب هي بالطباعة على لوحة المفاتيح.

يَألَف أغلب مستخدمي الحواسيب اليوم "واجهة المستخدم الرسومية" (GUI) فقط، وزرعت الشركات وبعض الأساتذة في عقولهم أن "واجهة سطر الأوامر" (CLI) هي شيءٌ مرعب أصبح من الماضي. وهذا الكلام غير صحيح البتة، لأن سطر الأوامر هو الطريقة التعبيرية الأفضل للتواصل مع الحاسوب، ويُمثّل نفس التأثير الذي تُحدِثه الكلمات في البشر. قد قيل بأن "الواجهة الرسومية تجعل المهمات السهلة أسهل، بينما سطر الأوامر يجعل المهمات الصعبة مُمكنة" ويبقى هذا الكلام صحيحًا إلى يومنا هذا.

ولأن لينُكس قد أُنشِئ على غرار بقية الأنظمة التي تنتمي إلى عائلة يونكس؛ فهو يحتوي على عددٍ ضخم من أدوات سطر الأوامر كما في يونكس الذي بدأ يشتهر في أوائل 1980 (على الرغم من أن تطويره قد تم قبل عقد من الزمن قبل ذلك التاريخ) وذلك قبل الانتشار والتأقلم مع الواجهات الرسومية. وبالنتيجة، طُوِّرت واجهة سطر أوامر غنية جدًا. وفي الحقيقة، أحد أهم الأسباب الذي جعلت المستخدمين الأوائل للينُكس يستخدمونه بدلًا من Window NT هو سطر الأوامر الذي جعل "المهمات الصعبة مُمكنةً".

عن ماذا يدور هذا الكتاب

يعطيك هذا الكتاب نظرةً واسعةً عن "العيش" في واجهة سطر الأوامر الخاصة بلينُكس. وعلى النقيض من الكتب التي تُركّز على برنامجٍ واحد، كصدفة bash. سيحاول هذا الكتاب جعلك تتأقلم مع واجهة سطر الأوامر، كيف تعمل؟ ما الذي يُمكنها فعله؟ ما هي أفضل طريقة لاستخدمها؟

هذا الكتاب ليس عن إدارة أنظمة لينُكس. على الرغم من أن أي حديث جدّي عن سطر الأوامر سيقود حتمًا إلى مواضيع تتعلق بإدارة الأنظمة؛ لكن هذا الكتاب ليس عنها. إلا أنه سيتطرق إلى بعض الاستخدامات الإدارية. وسيُهَيّء القارئ لأية دراسة إضافية لإدارة الأنظمة بتوفير بنية صلبة عن كيفية استخدام سطر الأوامر الذي يُمثل أداةً مُهمةً لأي مدير أنظمة.

يتوجه هذا الكتاب إلى مستخدمي نظام لينُكس. تُحاول العديد من الكتب تضمين الأنظمة الأُخرى في الشرح كنظام يونكس الأصلي ونظام Mac OS X، حيث يشرحون المواضيع العامة التي تتوافق مع تلك الأنظمة. يشرح هذا الكتاب توزيعات لينُكس الحديثة. لكن خمسًا وتسعين بالمائة من محتوى هذا الكتاب مُفيدٌ لمُستخدمي الأنظمة الأُخرى الشبيهة بِيونكس.

من يجب عليه قراءة هذا الكتاب

هذا الكتاب لمُستخدمي لينُكس الجدد الذين هاجروا من منصات أُخرى، أو للمستخدمين المبتدئين في سطر الأوامر. أغلب الظن أنك مستخدم متقدم لأحد إصدارات مايكروسوفت ويندوز. ربما رئيسك في العمل قد طلب منك إدارة خادم لينُكس. أو قد تكون مُستخدمًا عاديًا تَعِبَ من المشاكل الأمنية وأردت تجربة لينُكس.

تعلم سطر الأوامر ليس أمرًا سهلًا ويأخذ الكثير من الجهد والتعب. هو ليس صعبًا أو مستحيلًا، بل هو واسع

وضخم. يحتوي نظام لينُكس العادي على آلاف البرامج التي تستطيع استخدامها في سطر الأوامر. ها أنا ذا أُحذّرك، تعلم سطر الأوامر ليس بالمهمة السهلة.

على الجانب الآخر، سيؤتي تعلم سطر الأوامر أُكله. إذا كنت تظن الآن أنك مستخدم متقدم؛ فانتظر قليلًا، فأنت لا تعرف ما القوة الحقيقية بعد. وعلى النقيض من باقي مهارات الحاسوب، معرفة سطر الأوامر تدوم لفترة أطول. والمعلومات التي تتعلمها الآن ستبقى مُفيدةً بعد عشر سنوات على الأقل. لقد نجا سطر الأوامر من اختبار الزمن.

سنفرض أنه ليس لديك أيّة خبرة في البرمجة من قبل. لذا لا تقلق، سنبدأ ذاك المشوار معك أيضًا.

ما الذي يحتويه هذا الكتاب

أُختيرت المواضيع المقدَّمة في هذا الكتاب بعناية لكي تكون المعلومات فيها مُتسلسلة ومتعاقبة، كأن مدرسًا خاصًا يجلس جانبك لكي يُرشدك. أغلب المؤلفين يعاملون المحتوى معاملةً تصنيفيةً، مما يجعل ترتيب المحتوى معقولًا من وجهة نظر المؤلف؛ لكنه قد يربك القارئ.

هدف آخر هو تعريفك بطريقة التفكير الخاصة بِيونكس، التي تختلف عن طريقة التفكير في ويندوز. ستجد خلال مشوارنا بعض الملاحظات الجانبية التي تُساعدك على فهم طريقة عمل الأمور في لينُكس. لينُكس ليس مجرد برمجية فقط، بل هو أيضًا جزء من ثقافة يونكس.

ينقسم هذا الكتاب إلى أربعة أبواب، يشرح كل باب جانبًا من جوانب سطر الأوامر:

الباب الأول - أساسيات سطر الأوامر. نبدأ استكشافنا للغة الأساسية لسطر الأوامر. مع ذكر بنية الأوامر، والتنقل في نظام الملفات، وإجراء التعديلات باستخدام سطر الأوامر، والحصول على المساعدة والتوثيق للأوامر.

الباب الثاني - الإعدادات والبيئة يشرح تعديل ملفات الإعدادات للتحكم في سلوك النظام.

الباب الثالث - المهام الشائعة والأدوات الأساسية. نستكشف في هذا الباب العديد من المهمات الأساسية التي تُستخدم استخدامًا شائعًا من سطر الأوامر. تحتوي الأنظمة الشبيهة بِيونكس، كنظام لينُكس، العديد من برمجيات سطر الأوامر "التقليدية" التي تُستخدم لإجراء عمليات مهمة على البيانات.

الباب الرابع - كتابة سكربتات شِل يشرح كيفية كتابة سكربتات شِل (shell scripting). وهي تقنية تُستخدم لأتمتة أيّة مهمة شائعة.

كيف تقرأ هذا الكتاب

إذا كنت مُبتدئًا، فابدأ من بداية الكتاب وأكمل حتى نهايته. فهو كتاب تعليمي وليس مرجعًا.

التجهيزات

يجب أن يكون لديك نظام لينُكس جاهز للعمل لكي تستطيع استخدام هذا الكتاب. تستطيع الحصول عليه بطريقتين:

1. ثبّت لينُكس على حاسوبك (ليس من الضروري أن يكون حديثًا). ليس من المهم أي توزيعة قد اخترت، على الرغم من أن أغلب الأشخاص يستخدمون أوبنتو أو فيدورا أو أوبن سوزي. إذا كنت مُحتارًا فعليك بتجربة أوبنتو أولًا. قد يكون تثبيت توزيعة لينُكس سهلًا جدًا أو صعبًا جدًا وذلك وفقًا لعتاد حاسوبك وتوافقية التوزيعة معه.
2. استخدم القرص الحي. أحد الأشياء الرائعة التي تستطيع القيام بها مع العديد من توزيعات لينُكس هي تشغيلها مباشرةً من القرص المضغوط دون الحاجة إلى تثبيتها. عليك أن تُعدّل إعدادات `BIOS` في جهازك لجعله يُقلِع من القرص المضغوط ثم ضَع القرص المضغوط في محرك الأقراص وأعد إقلاع الجهاز. استخدام القرص الحي هو طريقة ممتازة لاختبار توافقية الحاسوب مع لينُكس قبل التثبيت. الجانب السلبي من استخدام القرص الحي هو أنه قد يكون أبطأ بكثير بالمقارنة مع وجود لينُكس مثبتًا على القرص الصلب. أوبنتو وفيدورا توفران القرص الحي بالإضافة إلى عدد كبير من التوزيعات الأخرى.

مهما كانت طريقة تشغيلك للينُكس، فستحتاج إلى امتيازات المستخدم الجذر (`root`) لكي تستطيع إنجاز الدروس في هذا الكتاب.

بعد أن استطعت تشغيل لينُكس على جهازك، ابدأ بالقراءة واتّبع الدروس على جهازك. أغلب المحتوى هو محتوى عملي، لذا، اجلس على جهازك وابدأ بالطباعة!

لماذا لا أسميه "غنو/لينُكس"

من الصحيح تسمية نظام تشغيل لينُكس باسم "غنو/لينُكس"، المشكلة مع "لينُكس" أنه لا توجد طريقة صحيحة تمامًا لتسميته لأنه كُتِب من قِبل العديد من الأشخاص الذين يعملون في مشاريع برمجية ضخمة. تقنيًّا: لينُكس هو اسم نواة نظام التشغيل ولا شيء أكثر. بالطبع، إن النواة مهمة للغاية بالنسبة إلى نظام التشغيل، لكنها لا تكفي لإنشاء نظام تشغيل كامل.

ريتشارد ستالمان، الفيلسوف العبقري الذي أوجد حركة البرمجيات الحرة، وشَكَّل مشروع غنو، وكتب الإصدار الأول من مترجم `c` الخاص بمشروع غنو (`gcc`)، وأنشأ رخصة غنو العمومية (GPL)، إلخ. يصر على أن تسميه "غنو/لينُكس" لكي تعكس مشاركة مشروع غنو. وعلى الرغم من أن مشروع غنو يسبق نواة لينُكس، والمساهمون في المشروع يستحقون التقدير، لكن وضعهم في الاسم غير مُنصف للبقية

الذين قاموا بمشاركات كبيرة في البرمجيات الحرة. على الرغم من ذلك، أظن أن الاسم "لينُكس/غنو" هو أدق تقنيًا لأن النواة تُقلِع أولًا ومن ثم كل شيء يعمل بالاعتماد عليها.

في الاستخدام الشائع، تُشير الكلمة "لينُكس" إلى النواة وجميع البرمجيات الحرة ومفتوحة المصدر الموجودة في توزيعة لينُكس الاعتيادية؛ هذا يعني أنها تُشير إلى جميع مكونات نظام لينُكس وليس فقط أدوات مشروع غنو. ويبدو أن العاملين في سوق أنظمة التشغيل يُفضلون أن تكون أسماء أنظمة التشغيل كلمة واحدة كأنظمة DOS, Windows, MacOS, Solaris, Irix, AIX. لذلك اخترت طريقة التسمية الأشهر. إذا كنت تفضل استخدام "غنو/لينُكس"، رجاءً أجرِ عملية بحث واستبدال ذهنية وأنت تقرأ هذا الكتاب.

تُرِكَتْ هَذِهِ الصَفْحَةُ فَارِغَةً عَمْدًا

الباب الأول:
أساسيات سطر الأوامر

الفصل الأول:

ما هي الصَدَفة؟

عندما نتكلم عن سطر الأوامر، نُشير إلى ما يُسمى "الصدفة" (`shell`). الصدفة هي برنامج يتلقى التعليمات والأوامر من لوحة المفاتيح ويُمررها إلى نظام التشغيل لكي ينفذ مهمةً معينةً. تُوفر جميع توزيعات لينُكس تقريبًا صدفة من مشروع غنو تسمى bash التي يُمثل اسمها اختصارًا للعبارة "Bourne Again SHell". التي تُشير إلى أن bash هي بديل مطور من sh، وهي الصدفة الأصلية الموجودة في أنظمة يونكس والتي كتبها Steve Bourne.

محاكيات الطرفية

نحتاج عند استخدام واجهة رسومية إلى برنامج نسميه "محاكي الطرفية" للتفاعل مع الصدفة. ربما ستجد واحدًا إذا بحثتَ في القوائم التي توفرها بيئة سطح المكتب لديك. كدي تستخدم `konsole` بينما غنوم تستخدم `gnome-terminal`. على الرغم من ذلك، غالبًا ما تُطلق كلمة `terminal` في القوائم. هنالك العديد من محاكيات الطرفية الأخرى متوفرة لنظام لينُكس؛ لكن جميع تلك المحاكيات توفر شيئًا واحدًا أساسيًا هو الوصول إلى الصدفة. ربما تُخصّص محاكي الطرفية الذي تستخدمه مُعتمدًا على الخيارات التي يوفرها.

ملاحظة: محاكي الطرفية الخاص بواجهة غنوم لا يدعم اللغة العربية. أي أن الأحرف العربية ستظهر مُتَقطِعَةً وبالمقلوب. لذا، يُنصح باستخدام المحاكي `Mlterm` (الذي يدعم العربية). لاحظ أن `konsole` يدعم العربية دعمًا جيدًا دون مشاكل في العرض.

أُولى خطواتك

فلنبدأ رحلتنا مع سطر الأوامر، افتح محاكي الطرفية، وعندما يظهر محاكي الطرفية سوف تشاهد عبارة شبيهة بالعبارة التالية:

```
[me@linuxbox ~]$
```

العبارة السابقة تسمى مِحث الصدفة (shell prompt) وتظهر عندما تكون الصدفة جاهزة لاستقبال المدخلات (input). وعلى الرغم من أن العبارة السابقة تختلف من توزيعة لأُخرى؛ لكنها غالبًا ما تحتوي

الجزء username@machinename يلحقه اسم مجلد العمل الحالي (سنناقشه لاحقًا) وإشارة الدولار.

إذا كان آخر حرف من مِحث الصدفة هو رمز المربع ("#") عوضًا عن إشارة الدولار؛ فهذا يعني أن جلسة الطرفية الحالية لها امتيازات الجذر، وهذا يعني أننا إما سجلنا دخولنا إلى الطرفية بحساب الجذر (root) أو أننا اخترنا محاكي طرفية يوفر امتيازات الجذر.

على فرض أن جميع الأمور سارت على ما يرام حتى الآن، فلنجرب طباعة بعض الحروف وإن لم تكن ذات مغزى:

```
[me@linuxbox ~]$ kaekfjaeifj
```

ولأن الأمر السابق ليس ذا معنى، فستخبرنا الصدفة بذلك، وستوفر مِحث الصدفة مرة أُخرى:

```
bash: kaekfjaeifj: command not found
[me@linuxbox ~]$
```

تأريخ الأوامر

إذا ضغطنا زر السهم العلوي، سنجد أن الأمر السابق "kaekfjaeifj" قد ظهر من جديد بعد مِحث الصدفة. وهذا ما يُسمى بتأريخ الأوامر (command history). تسجل معظم توزيعات لينُكس آخر خمسمائة أمر في التأريخ افتراضيًا. اضغط على زر السهم السفلي وستجد أن الأمر قد اختفى.

تحريك المؤشر

أعد استدعاء الأمر السابق بالضغط على زر السطر العلوي. جرب الآن الضغط على زري السهمين الأيمن والأيسر، لاحظ كيف تستطيع تغيير مكان المؤشر إلى أي موضع في الأمر. تُسهل هذه الميزة عملية تعديل الأوامر كثيرًا.

بعض النقاط حول استخدام الفأرة

على الرغم من أن الصدفة متعلقة بشكل أساسي بلوحة المفاتيح، لكنك تستطيع استخدام الفأرة في محاكي الطرفية. هنالك آلية في خادم العرض X (المحرك الذي يقف وراء تشغيل التطبيقات الرسومية) تسمح باستخدام النسخ واللصق "السريعَين". إذا حددت جزءًا من النص بالضغط على النص بالزر الأيسر وتمرير الفأرة فوقه (أو بالنقر المزدوج على كلمة لتحديدها)، فيُنسَخ إلى حافظة يديرها خادم X. وعند النقر بالزر الأوسط للفأرة، يُلصَق النص المُحدد في موضع المؤشر.

ملاحظة: لا تجرب استخدام `Ctrl-c Ctrl-v` لإجراء عمليات النسخ واللصق داخل نافذة الطرفية لأنهما لا يقومان بالنسخ واللصق بل يدرجان ما يُسمى "أكواد التحكم" التي تحمل معانٍ مختلفة عن النسخ واللصق بالنسبة للصدفة، وقد أُعتُمِدت هذه الأكواد قبل سنوات من ظهور نظام مايكروسوفت ويندوز.

بيئة سطح المكتب الخاصة بك (أغلب الظن كدي أو غنوم) تسعى إلى جعل سلوكها شبيهًا بسلوك ويندوز. غالبًا ما تكون آلية تنشيط النوافذ (focus policy) هي "انقر للتنشيط"، وهذا السلوك يناقض السلوك الافتراضي لخادم X: "التركيز (أو التنشيط) يتبع الفأرة"، هذا يعني أن النافذة ستُنَشَّط بمجرد مرور الفأرة فوقها (لكن لا يعني أن النافذة ستُنقَل إلى الأمامية) أي أنها ستقبل المدخلات. ضبط آلية التنشيط إلى "التركيز يتبع الفأرة" ستجعل النسخ واللصق مفيدًا وعمليًا أكثر. أظن أنك لو أعطيت هذا السلوك فرصةً كافيةً فإنك ستعتاد عليه وتجده مفيدًا. يمكنك تغيير هذا السلوك من الإعدادات الخاصة بمدير النوافذ الذي تستخدمه.

جرب بعض الأوامر البسيطة

الآن، وبعد أن تعلمنا كيفية الطباعة في الطرفية، لنجرب بعض الأوامر البسيطة. أول الأوامر التي سنجربها هو الأمر `date`، يُظهر هذا الأمر الوقت والتاريخ الحاليَين:

```
[me@linuxbox ~]$ date
Thu Oct 25 13:51:54 EDT 2007
```

أمر آخر شبيه بالأمر السابق هو `cal`، الذي يعرض افتراضيًا تقويم الشهر الحالي:

```
[me@linuxbox ~]$ cal
   October 2007
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

ولمعرفة مقدار الحجم التخزيني الفارغ في القرص الصلب، اطبع الأمر `df`:

```
[me@linuxbox ~]$ df
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sda2       15115452  5012392   9949716  34% /
/dev/sda5       59631908 26545424  30008432  47% /home
/dev/sda1         147764    17370    122765  13% /boot
tmpfs             256856        0    256856   0% /dev/shm
```

وبشكل مشابه، لمعرفة مقدار ذاكرة الوصول العشوائي غير المستخدمة في حاسوبك، استخدم الأمر free:

```
[me@linuxbox ~]$ free
         total       used         free      shared    buffers      cached
Mem:    513712     503976         9736           0       5312      122916
-/+ buffers/cache  375748       137964
Swap:  1052248     104712       947536
```

إنهاء جلسة الطرفية

بإمكانك إنهاء جلسة الطرفية إما بإغلاق نافذة محاكي الطرفية أو بطباعة الأمر exit في المِحث:

```
[me@linuxbox ~]$ exit
```

الطرفية المُختبئة خلف الستار

حتى وإن لم يكن لديك حاليًا أي محاكي للطرفية يعمل؛ فتوجد هنالك عدّة جلسات للطرفية تعمل خلف الواجهة الرسومية تسمى "الطرفيات الوهمية" (virtual terminals أو virtual consoles). يمكن الوصول لهذه الجلسات في أغلب توزيعات لينُكس بالضغط على Ctrl-Alt-F1 إلى Ctrl-Alt-F6. عندما يُبدّل إلى إحدى تلك الجلسات، ستوفر لك الجلسة ما يُسمى "مِحث الدخول" الذي تستطيع عبره كتابة اسم المستخدم وكلمة المرور. اضغط على Alt و F1-F6 للتبديل ما بين طرفية وهمية وأُخرى. للعودة إلى الواجهة الرسومية اضغط على Alt-F7.

الخلاصة

لقد تعرفنا في بداية مشورانا على الصدفة، وتعرّفنا على سطر الأوامر لأول مرّة، وتعلمنا كيف نبدأ ونُنهي

جلسة الطرفية. شاهدنا أيضًا كيف يمكن استخدام بعض الأوامر البسيطة، وطبّقنا القليل من تعديلات سطر الأوامر. هل كان ذلك مرعِبًا؟

الفصل الثاني:

الإبحار في نظام الملفات

أولى الأشياء التي علينا تعلمها (باستثناء تعلم طباعة الأوامر) هي كيفية الإبحار في نظام الملفات في لينُكس. سنناقش في هذا الفصل الأوامر الآتية:

- pwd – طباعة مسار مجلد العمل الحالي.
- cd – تغيير مجلد العمل الحالي.
- ls – عرض محتويات المجلد.

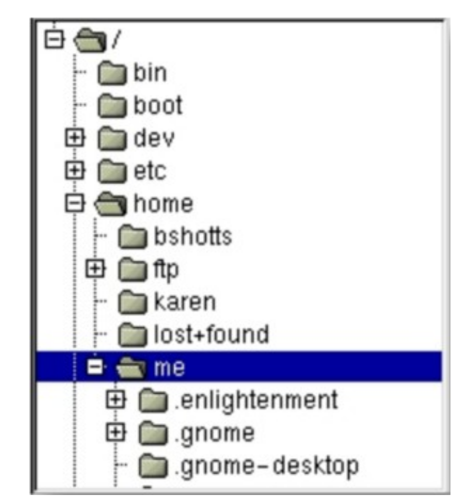
فهم شجرة نظام الملفات

كما في نظام ويندوز، تُنظم الأنظمة الشبيهة بِيونكس كنظام لينُكس ملفاتها بما يُسمى "هيكلة نظام الملفات". هذا يعني أن المجلدات والملفات مُنظّمة على شكل شجرة. المجلد الأول في نظام الملفات يُسمى المجلد الجذر. يحتوي المجلد الجذر على ملفاتٍ ومجلداتٍ فرعية، التي يمكن أن تحتوي على ملفات ومجلدات إضافية وهكذا.

وعلى النقيض من نظام ويندوز، الذي يعتمد على نظام ملفات منفصل لكل قرص تخزين، فإن الأنظمة الشبيهة بِيونكس كَالينُكس دائمًا ما تحتوي شجرة نظام ملفات وحيدة، دون الأخذ بعين الاعتبار طريقة أو عدد الأجهزة الملحقة بالحاسوب. يتم إضافة أقراص التخزين (أو بمعنى أدق، وصلها [mount]) في نقاط مختلفة في شجرة الملفات تحدد حسب رغبة مدير النظام، أو الأشخاص المسؤولين عن النظام.

مجلد العمل الحالي

يألف العديد منا مدراء الملفات الرسومية، التي تُمثِّل شجرة نظام الملفات كما في الشكل 1. لاحظ أن الشجرة تكون عادةً مقلوبةً رأسًا على عقب؛ حيث يكون المجلد الجذر إلى الأعلى ثم تتفرّع عنه باقي المجلدات.



الشكل 1: شجرة نظام الملفات كما يُظهِرها مدير الملفات الرسومي

لكن سطر الأوامر لا يحتوي على أيّة صور. لذا، سنحتاج إلى التفكير بطريقة مختلفة. لنتخيل أن نظام الملفات هو أشبه بشجرة مقلوبة نستطيع أن نقف في منتصفها على أحد أغصانها. نكون قادرين في أيّة لحظة على رؤية الملفات المحتواة في المجلد الحالي (أي أوراق الشجرة) والطريق إلى المجلد الذي يعلونا (يُسمى عادةً بالمجلد الأب) وأيّة مجلدات فرعية تقع تحتنا. المجلد الذي نقف فيه يُسمى مجلد العمل الحالي (current working directory). نستخدم الأمر pwd لكي نعرف مسار المجلد الذي نقف فيه:

```
[me@linuxbox ~]$ pwd
/home/me
```

عندما نسجل دخولنا إلى النظام (أو نبدأ جلسة محاكي الطرفية) يكون المجلد الحالي هو "المنزل" (home). يملك كل مستخدم مجلد منزل خاص به. عندما تكون مستخدمًا عاديًا (أي لا تملك امتيازات الجذر) فيكون مجلد المنزل هو المكان الوحيد الذي تستطيع الكتابة إلى الملفات فيه.

عرض قائمة بمحتويات المجلد الحالي

نستخدم الأمر ls لعرض قائمة بمحتويات المجلد الحالي:

```
[me@linuxbox ~]$ ls
Desktop Documents Music Pictures Public Templates Videos
```

في الواقع، نستطيع استخدام الأمر ls لعرض محتويات أي مجلد وليس فقط المجلد الحالي. هنالك العديد من الأمور الممتعة يمكنك فعلها أيضًا مع هذا الأمر. سنقضي المزيد من الوقت مع الأمر ls في الفصل القادم.

تغيير مجلد العمل الحالي

نستخدم الأمر cd لتغيير مجلد العمل الحالي (المكان الذي نقف فيه الآن في الشجرة المقلوبة). وذلك بكتابة الأمر cd يتبعه مسار المجلد الهدف الذي نريد الانتقال إليه. "المسار" هو الطريق الذي نسلكه عبر فروع شجرة نظام الملفات للوصول إلى المجلد الهدف. يمكن تحديد المسارات بطريقتَين: مسارات مطلقة أو مسارات نسبية. لنناقش المسارات المطلقة أولًا.

المسارات المطلقة

المسار المطلق هو المسار الذي يبدأ بالمجلد الجذر ويتبعه فروع شجرة نظام الملفات فرعًا فرعًا حتى نصل إلى المجلد (أو الملف) المطلوب. على سبيل المثال، هنالك مجلد في نظامك يحتوي على أغلب البرمجيات التي ثُبِّتَت على النظام؛ مسار ذاك المجلد هو usr/bin/. هذا يعني أنه ومن المجلد الجذر (يُمثَّل باستخدام الخط المائل في بداية المسار) هنالك مجلد يُسمى "usr" الذي بدوره يحتوي مجلدًا آخر يُسمى "bin".

```
[me@linuxbox ~]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
[me@linuxbox bin]$ ls


...Listing of many, many files ...
```

تستطيع ملاحظة كيف تغير المجلد الحالي إلى usr/bin/ والعدد الكبير من الملفات التي يحتويها هذا المجلد. لاحظ كيف تغير مِحث الصدفة. عمومًا، يُضمَّن اسم المجلد الحالي في المِحث ويتغير تلقائيًا عند تغيُّر المجلد.

المسارات النسبية

بينما يبدأ المسار المطلق من المجلد الجذر حتى المجلد الهدف، يبدأ المسار النسبي من المجلد الحالي. تَستخدم هذه الآلية رمزَين خاصَين لتمثيل المسارات النسبية في شجرة نظام الملفات، هذان الرمزان هما "." (نقطة واحدة) و ".." (نقطتَين متتاليتَين).

يُشير رمز النقطة "." إلى المجلد الحالي، ويُشير رمز النقطتَين المتتاليتَين ".." إلى المجلد الأب للمجلد الحالي.

الآن، لمعرفة آلية عمل المسارات النسبية، لنبدّل المجلد الحالي إلى /usr/bin:

```
[me@linuxbox ~]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

لنفرض أننا نريد أن ننتقل إلى المجلد الأب للمجلد /usr/bin الذي هو /usr. نستطيع القيام بذلك بطريقتَين. إما باستخدام المسارات المطلقة:

```
[me@linuxbox bin]$ cd /usr
[me@linuxbox usr]$ pwd
/usr
```

أو باستخدام المسارات النسبية:

```
[me@linuxbox bin]$ cd ..
[me@linuxbox usr]$ pwd
/usr
```

طريقتان مختلفتان تقومان بنفس العمل. أيهما نستخدم؟ سنستخدم بكل تأكيد الطريقة التي تتطلب طباعة أقل!

بشكل مُشابه، يمكننا التغيير من المجلد /usr إلى /usr/bin بطريقتَين مختلفتَين. إما بمسار مطلق:

```
[me@linuxbox usr]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

أو بمسار نسبي:

```
[me@linuxbox usr]$ cd ./bin
[me@linuxbox bin]$ pwd
/usr/bin
```

تجدر الإشارة إلى أنه باستطاعتك حذف "./" في أغلب الحالات، الأمر:

```
[me@linuxbox usr]$ cd bin
```

يقوم بنفس عمل الأمر السابق. عمومًا، إذا لم يُحَدَّد المسار لشيء ما، فسيؤخذ بعين الاعتبار مجلد العمل

الحالي.

بعض الاختصارات المفيدة

سنعرض بعض الطرق المفيدة لتغيير المجلد الحالي بسرعة في الجدول الآتي:

الجدول 2-1: اختصارات cd

الاختصار	النتيجة
`cd`	تغيير المجلد الحالي إلى المنزل الخاص بك.
`cd -`	تغيير مجلد العمل الحالي إلى مجلد العمل السابق.
`cd ~user_name`	تغيير المجلد الحالي إلى مجلد المستخدم user_name. على سبيل المثال، cd ~bob سيُغير المجلد إلى مجلد المنزل الخاص بالمُستخدم "bob".

حقائق مهمة حول أسماء الملفات

1. أسماء الملفات التي تبدأ بنقطة هي ملفات مخفيّة. هذا يعني أن أمر ls لن يعرض هذه الملفات إلا إذا استخدمتَ ls -a. ستتموضع عدّة ملفات مخفية في مجلد المنزل لتضبط بعض الإعدادات لحسابك عندما يُنشأ حسابك لأول مرة في نظام التشغيل. سوف نلقي نظرة عن كثب على بعض تلك الملفات في فصولٍ قادمة لكي تعرف كيف تخصص بيئة التشغيل لديك. بالإضافة إلى ذلك، تضع بعض البرمجيات ملفات الإعدادات الخاصة بها في مجلد المنزل كملفات مخفية.
2. أسماء الملفات والأوامر في لينُكس، كما في يونكس، هي حساسة لحالة الأحرف. الاسمَين "File1" و "file1" يُشيران إلى ملفَين مختلفَين تمامًا.
3. نظام لينُكس لا يأخذ بعين الاعتبار "امتداد الملف" كما في أنظمة التشغيل الأخرى. لذا، تستطيع تسمية الملفات كما يحلو لك. محتوى أو الهدف من الملف يُحدد بطرق أُخرى. وعلى الرغم من أن الأنظمة الشبيهة بِيونكس لا تستخدم امتداد الملف لتحديد محتواه أو الهدف منه، إلا أن بعض البرمجيات تقوم بذلك.
4. على الرغم من أن نظام لينُكس يدعم أسماء الملفات الطويلة والتي تحتوي على فراغات وعلامات ترقيم؛ لكن اقتصر على استخدام النقطة، والشرطة (-)، والشرطة السفلية (_) في

أسماء الملفات. أهم ما في الأمر هو عدم تضمين أي فراغ في أسماء الملفات. إذا أردت تمثيل الفراغات ما بين الكلمات في اسم الملف، فاستخدم الشرطة السفلية. ستعرف فائدة ذلك لاحقًا.

الخلاصة

لقد رأينا كيف تُعامِل الصدفة بنية المجلدات في النظام. تعلمنا الفروقات ما بين المسارات النسبية والمطلقة، وبعض الأوامر البسيطة التي تُستخدم للتنقل ما بين المجلدات. سنستخدم هذه المعلومات في الفصل القادم لبدء رحلتنا في نظام تشغيل لينُكس.

الفصل الثالث:

استكشاف النظام

حان الوقت الآن لكي نقوم برحلة منظمة في نظام لينُكس الخاص بنا بعد أن تعلمنا طريقة التنقل في نظام الملفات. لكن علينا، قبل البدء، تعلم الأوامر الأساسية التي ستساعدنا في ذلك:

- ls - عرض محتويات مجلد ما.
- file - تحديد نوع ملف ما.
- less - عرض محتويات ملف ما.

عرض قائمة بمحتوى مجلدٍ ما باستخدام ls

ربما يكون الأمر ls هو من أكثر الأوامر فائدةً وذلك لأنه يُمكّننا من مشاهدة محتوى مجلدٍ ما وإظهار عدد من خاصيات الملفات والمجلدات المهمة. يُمكننا ببساطة استخدام الأمر ls لعرض قائمة بمحتويات مجلد العمل الحالي من ملفاتٍ ومجلداتٍ فرعية:

```
[me@linuxbox ~]$ ls
Desktop Documents Music Pictures Public Templates Videos
```

علاوةً على استخدامه لعرض محتويات المجلد الحالي، يمكن تحديد المجلد المُراد عرض محتوياته بتمرير مساره كوسيط (Argument) للأمر ls كما في المثال الآتي:

```
[me@linuxbox ~]$ ls /usr
bin games   kerberos libexec sbin  src
etc include lib      local   share tmp
```

ويُمكننا أيضًا تمرير أكثر من مجلد كوسائط. سنعرض في هذا المثال محتويات مجلد المنزل (الذي يُرمز له بالرمز "~") والمجلد usr/:

```
[me@linuxbox ~]$ ls ~ /usr
/home/me:
Desktop Documents Music Pictures Public Templates Videos
```

```
/usr:
bin   games    kerberos  libexec  sbin    src
etc   include  lib       local    share   tmp
```

نستطيع أيضًا تغيير صياغة المُخرجات لكي تُظهِر المزيد من التفاصيل:

```
[me@linuxbox ~]$ ls -l
total 56
drwxr-xr-x  2 me me  4096 2007-10-26 17:20 Desktop
drwxr-xr-x  2 me me  4096 2007-10-26 17:20 Documents
drwxr-xr-x  2 me me  4096 2007-10-26 17:20 Music
drwxr-xr-x  2 me me  4096 2007-10-26 17:20 Pictures
drwxr-xr-x  2 me me  4096 2007-10-26 17:20 Public
drwxr-xr-x  2 me me  4096 2007-10-26 17:20 Templates
drwxr-xr-x  2 me me  4096 2007-10-26 17:20 Videos
```

غيرنا صيغة المخرجات إلى النمط الطويل أو التفصيلي بإضافة الخيار "l-" إلى الأمر ls.

الخيارات والوسائط

عادةً ما يُتبَع الأمر بخيار واحد أو أكثر كي يُعدِّل سلوكه، ويأتي بعده وسيط واحد أو أكثر محددًا "الأشياء" التي سيُنفّذ الأمر عليها. لذا، يكون الشكل العام لأغلب الأوامر كالآتي:

```
command -options arguments
```

تتكون معظم خيارات الأوامر من حرف واحد مسبوق بشرطة. على سبيل المثال، "l-"؛ لكن أغلب الأوامر، بما فيها تلك التي أُنشئت من قِبل مشروع غنو، تدعم استخدام الخيارات الطويلة التي تتكون من كلمة مسبوقة بشرطتَين اثنتَين. ويدعم العديد من الأوامر كتابة الخيارات القصيرة ملتصقةً ببعضها. سنُمرر في المثال التالي خيارين قصيرين للأمر ls: الخيار "l" الذي يعرض المخرجات بالصيغة التفصيلية، والخيار "t" الذي يرتب النتائج وفق تاريخ تعديل الملفات:

```
[me@linuxbox ~]$ ls -lt
```

سنضيف إلى الأمر السابق الخيار "reverse--" لعكس الترتيب الناتج:

```
[me@linuxbox ~]$ ls -lt --reverse
```

لاحظ أن خيارات الأوامر -كما هي أسماء الملفات في لينُكس- حساسة لحالة الأحرف.

لدى الأمر ls عدد كبير من الخيارات، يعرض الجدول الآتي أبرزها وأهمها:

الجدول 3-1: خيارات ls الشائعة

الخيار	الخيار الطويل	الشرح
-a	--all	عرض جميع الملفات الموجودة في المجلد بما فيها الملفات التي تبدأ بنقطة والتي لا تظهر افتراضيًا (الملفات المخفية).
-A	--almost-all	كالخيار a-، إلا أنه لا يعرض "." (المجلد الحالي) و ".." (المجلد الأب).
-d	--directory	سيعرض الأمر ls محتويات مجلدٍ ما إذا مُرِّرَ كوسيط ولا يعرض معلومات المجلد نفسه في غياب هذا الخيار. استخدم هذا الخيار (بالإضافة إلى الخيار "l-") لعرض معلومات عن المجلد بدلًا من عرض محتوياته.
-F	--classify	إظهار رمز خاص في نهاية كل قيد (ملف أو مجلد). على سبيل المثال: سيعرض"/" إذا كان القيد مجلدًا.
-h	--human-readable	عرض الحجم التخزيني للملفات عرضًا سهل القراءة للمستخدم بدلًا من عرضه بالبايتات وذلك عند استخدام طريقة العرض التفصيلية.
-l		عرض النتائج بطريقة العرض التفصيلية.
-r	--reverse	عرض النتائج بترتيب معكوس. افتراضيًا، يرتب الأمر ls النتائج حسب التسلسل الأبجدي.
-S		ترتيب النتائج حسب الحجم التخزيني للملف.
-t		ترتيب النتائج حسب تاريخ التعديل.

نظرة عن قُرب على طريقة العرض التفصيلية

كما شاهدنا سابقًا، إن الخيار "l-" سيجعل الأمر ls يُظهِر النتائج بصيغة العرض التفصيلية التي تحتوي على عدد كبير من المعلومات المفيدة، هذا هو ناتج تنفيذ الأمر ls مع الخيار "l-" في مجلد Examples في

توزيعة أوبنتو:

```
-rw-r--r--    1  root   root  3576296 2007-04-03 11:05 Experience ubuntu.ogg
-rw-r--r--    1  root   root  1186219 2007-04-03 11:05 kubuntu-leaflet.png
-rw-r--r--    1  root   root     47584 2007-04-03 11:05 logo-Edubuntu.png
-rw-r--r--    1  root   root     44355 2007-04-03 11:05 logo-Kubuntu.png
-rw-r--r--    1  root   root     34391 2007-04-03 11:05 logo-Ubuntu.png
-rw-r--r--    1  root   root     32059 2007-04-03 11:05 oo-cd-cover.odf
-rw-r--r--    1  root   root    159744 2007-04-03 11:05 oo-derivatives.doc
-rw-r--r--    1  root   root     27837 2007-04-03 11:05 oo-maxwell.odt
-rw-r--r--    1  root   root     98816 2007-04-03 11:05 oo-trig.xls
-rw-r--r--    1  root   root    453764 2007-04-03 11:05 oo-welcome.odt
-rw-r--r--    1  root   root    358374 2007-04-03 11:05 ubuntu Sax.ogg
```

لنلقِ نظرة على مختلف الحقول من أحد الملفات السابقة ولنشرح معناها:

الجدول 3-2: حقول طريقة العرض التفصيلية في ls

الحقل	الشرح
`-rw-r--r--`	أذونات الوصول إلى الملف. أول محرف يُمثل نوع القيد. الشرطة تعني أن القيد ما هو إلا ملف عادي، بينما "d" تعني أن القيد هو عبارة عن مجلد، وهكذا. المحارف الثلاثة التي تليها تُحدد أذونات الوصول إلى الملف بالنسبة إلى المستخدم المالك للملف. المحارف الثلاثة التي تليها تحدد أذونات الوصول إلى المجموعة المالكة. أما آخر ثلاثة محارف، فهي تحدد الأذونات لأي مُستخدم آخر. سيُناقش المعنى الكامل لهذه الأذونات في الفصل التاسع "الأذونات".
1	عدد الوصلات الصلبة للملف، راجع النقاش حول الوصلات في نهاية هذا الفصل.
`root`	اسم المستخدم المالك للملف.
`root`	اسم المجموعة المالكة للملف.
32059	حجم الملف التخزيني مُقدرًا بالبايت.
2007-04-03 11:05	الوقت والتاريخ لآخر عملية تعديل.
`oo-cd-cover.odf`	اسم الملف.

تحديد نوع الملف باستخدام الأمر file

من المفيد أن نعرف ما الذي يحتويه ملفٌ ما بينما نستكشف النظام. وذلك باستخدام الأمر file الذي يُحدد نوع الملف. وكما ناقشنا سابقًا، امتدادات الملفات في لينُكس غير ضرورية لتحديد محتوى الملف. على الرغم من أن ملفًا يحمل الاسم "picture.jpg" يُتوقع منه أن يحتوي على صورة مضغوطة بصيغة JPEG. لكن ذلك ليس ضروريًا في لينُكس. في العادة، نستخدم الأمر file على النحو الآتي:

```
file filename
```

يعرض الأمر file شرحًا موجزًا عن محتويات الملف. مثال:

```
[me@linuxbox ~]$ file picture.jpg
picture.jpg: JPEG image data, JFIF standard 1.01
```

هنالك العديد من أنواع الملفات. في الواقع، إحدى أشهر الأفكار في الأنظمة الشبيهة بِيونكس كَلينُكس تعتبر أن "كل شيء هو ملف". سترى مقدار صحة هذه العبارة خلال رحلتنا في فصول هذا الكتاب.

صحيح أن أغلب أنواع الملفات في نظامك هي أنواع مألوفة لديك، MP3 و JPEG على سبيل المثال، لكن يوجد العديد من الأنواع أقل شهرة بل بعضها غريب للغاية.

عرض محتويات الملفات باستخدام الأمر less

الأمر less هو برنامج يعرض الملفات النصية. يوجد في نظام لينُكس العديد من الملفات التي تحتوي على نصوص قابلة للقراءة؛ الأمر less يوفر طريقة جيدة للاطلاع على محتواها.

ما هو "النص"؟

هنالك العديد من الطرق لتمثيل المعلومات على الحاسوب. جميع الطرق تقوم بإيجاد علاقة تربط المعلومات وبعض الأرقام التي تُستخدم لتمثيلها. لا تستطيع الحواسيب فهم أي شيء عدا الأرقام؛ لذا، تُحوَّل جميع البيانات إلى أرقام.

بعض نظم تمثيل البيانات معقدة جدًا (كما في ملفات الفيديو المضغوطة)، بينما يكون بعضها الآخر بسيطًا للغاية. أحد أقدم وأبسط تلك النظم يدعى "نصوص ASCII". ASCII (تُلفظ "آس-كي") هي اختصار للعبارة الإنكليزية "American Standard Code for Information Interchange". التي هي آلية بسيطة أُستخدِمت لأول مرّة في الآلات الكاتبة لكي تربط الحروف الموجودة في لوحة المفاتيح

بالأرقام.

النص هو نمط ربط بسيط "واحد-إلى-واحد"، يربط المحارف مع الأرقام. الصيغة مضغوطة جدًا. حيث يتحول خمسون حرفًا من النص إلى خمسين بايتًا. من المهم فهم أن النص يحتوي على ربط المحارف إلى الأرقام فقط. وهو ليس كالمستندات التي تُنشأ باستخدام مايكروسوفت أوفيس وورد أو ليبر أوفيس رايتر. بالإضافة إلى احتواء تلك الملفات على نص ASCII بسيط، فهي تحتوي أيضًا على عناصر غير نصية تُستخدم لوصف بُنية وتنسيق المُستند. مستندات ASCII تحتوي فقط على المحارف أنفسهم وبعض أكواد التحكم كمسافة الجدولة (tab) وعلامات السطر الجديد.

خُزِّنَت العديد من الملفات في نظام لينُكس على شكل ملفات نصية يسهل التعامل معها بباقي الأدوات التي تعالج النصوص. وحتى نظام ويندوز يُدرك هذه الصيغة. البرنامج الشهير NOTEPAD.EXE هو محرر لملفات ASCII النصية.

لماذا نريد أن نعرف محتوى الملفات النصية؟ السبب هو أن أغلب ملفات الإعدادات التي يستخدمها النظام مخزنة بهذه الصيغة، وتزيد إمكانية قراءة هذه الملفات من فهمنا لكيفية عمل نظام التشغيل. بالإضافة إلى ذلك، تُخَزَّن العديد من البرمجيات التي يستخدمها النظام (تسمى سكربتات scripts) بهذه الصيغة. سنتعلم في الفصول الأخيرة من هذا الكتاب طريقة تعديل ملفات الإعدادات لتغيير سلوك النظام بالإضافة إلى كتابة السكربتات الخاصة بنا، لكن حاليًا كل ما نريد فعله هو إلقاء نظرة على محتواها.

يُستخدم الأمر less على النحو الآتي:

```
less filename
```

سيسمح لك برنامج less بالتمرير إلى الأمام وإلى الخلف في الملفات النصية. على سبيل المثال، نستخدم الأمر الآتي لكي نتطلع على جميع حسابات مستخدمي النظام:

```
[me@linuxbox ~]$ less /etc/passwd
```

تستطيع الآن مشاهدة محتوى الملف بعد بدء برنامج less؛ وفي حال كان الملف لا يتسع في صفحة واحدة، فتستطيع التمرير إلى الأعلى والأسفل. اطبع الحرف "q" لإغلاق less.

يسرد الجدول الآتي أبرز اختصارات لوحة المفاتيح التي يستخدمها less:

الجدول 3-3: أوامر less

الزر	الشرح
Page Up أو الحرف b	التمرير إلى الصفحة السابقة.
Page Down أو الفراغ	التمرير إلى الصفحة التالية.
السهم العلوي	التمرير إلى السطر السابق.
السهم السفلي	التمرير إلى السطر التالي.
G	التمرير إلى نهاية المستند.
1G أو g	التمرير إلى بداية المستند.
/characters	البحث عن مطابقة للمحارف "characters".
n	عرض المطابقة التالية للبحث السابق.
h	عرض شاشة المساعدة
q	الخروج من less.

less is more

طُوِّرَ البرنامج less لكي يكون بديلًا مُحسنًا عن البرنامج more الموجود في يونكس. الكلمة "less" هي مجرد تلاعب بالألفاظ في عبارة "less is more" (عبارة يستخدمها المعماريون والمصممون المعاصرون).

صُنِّف less بين البرامج على أنه "قارئ صفحات" (pager)، وهي برمجيات تسمح بعرض الملفات النصية الطويلة على شكل صفحات متتالية. بينما كان برنامج more يسمح فقط بالانتقال إلى الأمام، يسمح less بالانتقال إلى الأمام والخلف، بالإضافة إلى عددٍ كبيرٍ من الميزات الأخرى.

رحلة في مجلدات نظام التشغيل

بنية نظام الملفات في لينُكس تشبه إلى حدٍ بعيد تلك الموجودة في بقية الأنظمة الشبيهة بِيونكس. في الواقع،

نُشِرت بنية نظام الملفات على شكل معيار يُطلق عليه "معيار هيكلة نظام الملفات" (Linux Filesystem Hierarchy Standard). لا تتَّبع جميع التوزيعات هذا المعيار اتّباعًا كاملًا ودقيقًا، إلا أن بنية نظام الملفات الذي تعتمده مختلف التوزيعات لا يختلف إلا ببعض الأشياء البسيطة جدًا.

سنبدأ الآن بالتنقل في نظام لينُكس الخاص بنا ومعرفة كيف يعمل. وسنحصل على فرصة للتدرب على مهارات التنقل في النظام. أحد الأشياء التي سنكتشفها هي أن أغلب الملفات المهمة تتكون من مجرد نص بسيط قابل للقراءة. جرّب الأوامر التالية أثناء القيام بهذه الرحلة:

1. انتقل إلى المجلد باستخدام الأمر `cd`.
2. اعرض محتوى المجلد بالأمر `ls -l`.
3. حدد نوع أحد الملفات إذا وجدته مثيرًا للاهتمام.
4. إذا شككت بأن محتوى الملف نصي، فاعرض محتواه باستخدام `less`.

تذكر خدعة النسخ واللصق! يمكنك تحديد اسم الملف بالنقر المزدوج عليه بالفأرة (إن كنت تستخدمها) ومن ثم الصقه بالنقر على الزر الأوسط للفأرة.

لا تخف من الاطلاع على أي شيء عندما تتجول في نظام الملفات. يخاف المُستخدمون العاديون عادةً من "تخريب النظام ". إذا اشتكى أي أمر من مشكلة فدعه وانتقل إلى شيء آخر. اقضِ بعض الوقت في استكشاف النظام. تذكر أنه لا توجد أيّة أسرار في لينُكس!

يعرض الجدول الآتي بعض المجلدات التي يُمكننا استكشافها:

الجدول 3-4: المجلدات الموجودة في أنظمة لينُكس

المجلد	الشرح
`/`	المجلد الجذر. يبدأ كل شيء من هنا.
`/bin`	يحتوي على الملفات الثنائية (binaries) للبرمجيات التي يجب أن تتوفر لكي يقلع ويعمل النظام.
`/boot`	يحتوي على نواة لينُكس، وصورة قرص الذاكرة العشوائية الابتدائي (للأقراص الضرورية في وقت الإقلاع)، ومُحمّل الإقلاع (boot loader). الملفات التي تستحق الانتباه: • الملفان `/boot/grub/grub.conf` و `menu.lst`، اللذان يحتويان على الإعدادات التي يستخدمها محمل الإقلاع.

	• ملف `/boot/vmlinuz` الذي يُمثّل نواة لينُكس.
`/dev`	هذا المجلد الخاص يحتوي على "عقد الأجهزة" (device nodes). العبارة "كل شيء ملف" تنطبق أيضًا على الأجهزة. تُبقي النواة قائمةً بجميع الأجهزة التي تستطيع التعامل معها في هذا المجلد.
`/etc`	يحتوي المجلد `/etc` على جميع ملفات الإعدادات التي تُطَبَّق على كامل النظام. ويحتوي أيضًا على مجموعة من سكربتات الشِل (shell scripts) التي يبدأ كلٌ منها خدمةً من خدمات النظام في وقت الإقلاع. أغلب الملفات الموجودة في هذا المجلد هي ملفات نصية عادية. الملفات التي تستحق الانتباه: جميع الملفات في هذا المجلد تستحق الانتباه، هذه قائمة صغيرة منها: • الملف `/etc/crontab`: يحتوي هذا الملف على مواعيد تشغيل المهام الدورية. • الملف `/etc/fstab`: جدول يحتوي على قائمة بأجهزة التخزين وما يُقابلها من نقاط الوصل. • الملف `/etc/passwd`: يحتوي على قائمة بجميع حسابات المستخدمين.
`/home`	عادةً، يُمنح كل مُستخدم من مستخدمي النظام مجلدًا في `/home`. المُستخدمون العاديون لا يملكون امتياز الكتابة على الملفات إلا في مجلد المنزل الخاص بهم. هذا التقييد يحد من إمكانية تخريب النظام بسبب خطأ من جانب المستخدم العادي.
`/lib`	يحتوي على المكتبات البرمجية المشتركة بين برمجيات النظام الأساسية، هذه الملفات شبيهة بملفات `DLL` في نظام ويندوز.
`/lost+found`	يحتوي كل قطاع مُهيئ بنظام ملفات لينُكس (مثلًا، نظام الملفات `ext3`) على هذا المجلد. يُستخدم هذا المجلد بعد استرجاع جزئي لنظام الملفات بعد تعرضه للتلف. سيبقى هذا المجلد فارغًا إذا لم يتعرض نظامك إلى مشكلة في نظام الملفات.
`/media`	يحتوي هذا المجلد في توزيعات لينُكس الحديثة على نقاط الوصل للأقراص القابلة للإزالة كأجهزة USB أو `CD-ROM` ...إلخ. التي تُوصَل مباشرةً عند وضعها في الحاسوب.

/mnt	يحتوي المجلد mnt/ على نقاط الوصل للأجهزة القابلة للإزالة التي وُصِلَت يدويًا وذلك في توزيعات لينُكس القديمة.
/opt	يُستخدم المجلد opt/ لتثبيت البرمجيات الاختيارية (optional). عادةً ما يُستخدم هذا المجلد لتخزين ملفات البرمجيات التجارية التي ثُبِّتت على نظامك.
/proc	المجلد proc/ هو مجلد من نوع خاص. فهو عبارة عن نظام ملفات وهمي يُدار من قِبل نواة لينُكس وليس عبارة عن ملفات موجودة على قرصك الصلب. يمكن اعتبار "الملفات" الموجودة فيه أنها "ثقوب" أو "فجوات" تؤدي إلى النواة. الملفات الموجودة في هذا المجلد قابلة للقراءة وتعطيك فكرة عن آلية تعامل النواة مع حاسوبك.
/root	هذا المجلد هو مجلد المنزل للمستخدم الجذر.
/sbin	يحتوي هذا المجلد على الملفات الثنائية الخاصة بالنظام. هذه البرمجيات تقوم بأعمال مهمة للنظام وتُستخدم عادةً من قِبل المستخدم الجذر فقط.
/tmp	يُخزِّن المجلد tmp/ الملفات المؤقتة التي أنشأتها مختلف البرامج. بعض الأنظمة تكون مضبوطة بأن تمسح جميع محتويات هذا المجلد في كل مرة يُعاد فيها إقلاع النظام.
/usr	غالبًا ما يكون المجلد usr/ أكبر المجلدات في نظام لينُكس، لأنه يحتوي على جميع البرامج وملفات الدعم التي يستعملها المستخدمون العاديون.
/usr/bin	يحتوي المجلد usr/bin/ على الملفات التنفيذية للبرامج المثبتة في نظام لينُكس الخاص بك. من الشائع أن يحتوي هذا المجلد على آلاف البرامج القابلة للاستخدام.
/usr/lib	يحتوي هذا المجلد على المكتبات المشتركة بين البرمجيات الموجودة في المجلد usr/bin/.
/usr/local	شجرة الملفات المحتواة في المجلد usr/local/ هي المكان الذي تُخزّن فيه ملفات البرمجيات غير المُضمنة افتراضيًا مع توزيعتك. البرامج المبنية من المصدر توضع افتراضيًّا في المجلد usr/local/bin/. يكون ذاك المجلد فارغًا في الأنظمة المُثبّتة حديثًا حتى يُقرر مدير النظام أن يضع فيه شيئًا ما.

/usr/sbin	يحتوي على برمجيات إضافية لإدارة النظام.
/usr/share	يحتوي المجلد usr/share/ على جميع البيانات المشتركة بين البرامج الموجودة في usr/bin/، بما فيها ملفات الإعدادات الافتراضية والأيقونات وخلفيات الشاشة والملفات الصوتية ...إلخ.
/usr/share/doc	أغلب الحزم المثبتة على نظامك تحتوي على توثيق يُبيّن طريقة استخدمها. سنجد ملفات التوثيق منظمةً حسب الحزم في المجلد usr/share/doc/.
/var	يوجد في أغلب المجلدات التي ناقشناها مُسبقًا باستثناء home/ و tmp/ محتوى ثابت نسبيًا. هذا يعني أن محتواها لا يتغير كثيرًا. شجرة الملفات الموجودة في المجلد var/ تحتوي على البيانات التي يمكن أن يتغير محتواها دوريًّا. على سبيل المثال، قواعد البيانات المختلفة، ملفات البريد الإلكتروني ...إلخ.
/var/log	يحتوي var/log/ على الملفات التي تحتوي "السجلات" (log files). هذه الملفات مهمة جدًا ويجب أن نطلع عليها بين الحين والآخر. أحد أهم تلك الملفات هو var/log/messages/. لاحظ أنك تحتاج، ولأسبابٍ أمنية، إلى امتيازات الجذر لمشاهدة محتوى بعض هذه الملفات.

الوصلات الرمزية

ربما نواجه بعض الملفات عند استكشاف محتوى المجلدات بالشكل الآتي:

```
lrwxrwxrwx 1 root root 11 2007-08-11 07:34 libc.so.6 -> libc-2.6.so
```

لاحظ أن أول حرف في القيد هو "l"، ويبدو أيضًا أن لدى القيد اسمَين! هذا هو نوع خاص من الملفات يُسمى بالوصلة الرمزية (symbolic link أو soft link أو symlink). يُسمح في الأنظمة الشبيهة بِيونكس بأن يُشار إلى الملف بأكثر من اسم واحد. قد لا يبدو الأمر مفيدًا للغاية، لكنه كذلك!

تخيل هذا السيناريو: أحد البرامج يستخدم موردًا مشتركًا وليكن اسمه "foo" لكن إصدارات "foo" تتغير باستمرار. وبالطبع يكون من الجيد أن يحتوي اسم الملف على رقم الإصدار كي يتمكن مدير النظام (أو أي جهة أُخرى) من معرفة أي إصدار من "foo" قد ثُبِّت. وهذا ما يُسبب المشكلة الآتية، إذا غيرنا اسم الملف فإننا سنحتاج إلى تغييره أيضًا في كل برنامج يستخدمه وذلك في كل مرة نقوم فيها بتحديث ذاك الملف. لا يبدو الأمر مسليًا أبدًا!

لنفترض أننا ثبتنا الإصدار 2.6 من "foo" الذي يحمل الاسم "foo-2.6" ومن ثم أنشأنا وصلةً رمزيةً تُشير لذاك الملف باسم "foo". هذا يعني أنه عندما يستخدم أحد البرامج "foo" فإنه في الواقع يستخدم "foo-2.6". البرامج التي تعتمد على "foo" تستطيع استخدامه وفي الوقت نفسه نستطيع معرفة أي إصدار من "foo" قد ثُبِّت. وعندما يحين الوقت لكي نحدث إلى "foo-2.7"، نحذف الوصلة "foo" التي تُشير إلى الإصدار القديم، وننشئ وصلةً جديدةً تُشير إلى الإصدار الجديد. هذا الأمر لا يحل مشكلة تحديث النسخ فحسب، بل يسمح لنا بالاحتفاظ بالإصدارَين. لنفترض أن الإصدار "foo-2.7" يحتوي على علّة ما، ونحن نحتاج إلى استعادة النسخة القديمة، فإننا نحذف بكل بساطة الوصلة "foo" التي تُشير إلى الإصدار الجديد ونُعيد ربطها مع الإصدار القديم.

القيد الموجود في الأعلى (من مجلد lib/ في توزيعة فيدورا) يُظهر أن الوصلة الرمزية التي تدعى "libc.so.6" تُشير إلى مكتبة "libc-2.6.so". هذا يعني أنه عندما يستخدم برنامج ما "libc.so.6" فإنه سيحصل على الملف "libc-2.6.so".

سنتعلم طريقة إنشاء الوصلات الرمزية في الفصل القادم.

الوصلات الصلبة

لأننا نتحدث عن موضوع الوصلات، فيجدر بنا ذكر النوع الثاني من الوصلات الذي يُسمى الوصلات الصلبة (hard links). تسمح الوصلات الصلبة (كما في الوصلات الرمزية) بأن نُشير إلى ملف بأسماء مختلفة، لكنها تفعل ذلك بطريقة مختلفة تمامًا. سنناقش بالتفصيل الفروق ما بين الوصلات الرمزية والوصلات الصلبة في الفصل القادم.

الخلاصة

لقد تعلمنا الكثير عن نظامنا بعد أن أنهينا رحلتنا المشوقة. لقد رأينا مختلف الملفات والمجلدات ومحتوياتهم. أهم ما يجب عليك تعلمه من هذه الرحلة هو أن نظام لينُكس هو نظام مفتوح قابل للتخصيص كثيرًا. توجد العديد من الملفات المهمة في لينُكس التي تكون مكتوبةً على شكل ملفات نصية بسيطة قابلة للقراءة. وعلى النقيض من باقي الأنظمة الاحتكارية، يوفّر نظام لينُكس كل شيء فيه للاطلاع والدراسة.

الفصل الرابع:

معالجة الملفات والمجلدات

الآن، وبعد تعلمنا للعديد من الأمور في الفصول السابقة؛ حان الوقت للعمل الجدي. سيشرح لك هذا الفصل الأوامر الآتية:

- cp – نسخ الملفات والمجلدات.
- mv – نقل أو إعادة تسمية الملفات والمجلدات.
- mkdir – إنشاء المجلدات.
- rm – حذف الملفات والمجلدات.
- ln – إنشاء وصلات صلبة ورمزية.

الأوامر الخمسة السابقة من أكثر الأوامر استخدامًا في لينُكس. تُستخدم الأوامر السابقة لمعالجة الملفات والمجلدات على حدٍ سواء.

لكن منصفين: يمكن تنفيذ المهام السابقة بسهولة باستخدام مدير ملفات رسومي. يمكنك سحب وإفلات ملف من مجلد إلى آخر، قص ولصق الملفات وحذفهم ...إلخ. لماذا إذًا سنستخدم هذه الأوامر القديمة؟

الجواب هو في القوة والمرونة التي تتمتع بها هذه الأوامر. صحيح أن القيام بعمليات المعالجة البسيطة على الملفات يكون سهلًا للغاية في الواجهة الرسومية؛ لكن القيام بعمليات المعالجة المعقدة أسهل بكثير في سطر الأوامر. على سبيل المثال، كيف تستطيع نسخ جميع ملفات HTML الموجودة في مجلد ما إلى مجلد آخر، لكن فقط نسخ الملفات التي لا توجد في المجلد الهدف أو استبدال النسخة الأقدم بالنسخة الأحدث من الملف؟ القيام بذلك صعب جدًا باستخدام مدير ملفات رسومي، لكنه سهل جدًا في سطر الأوامر:

```
cp -u *.html destination
```

المحارف البديلة

لنتعرف قبل الخوض في التفاصيل على ميزة من ميزات الصدفة التي تجعلها فعالة لدرجة كبيرة. لما كانت الصدفة تستخدم أسماء الملفات بكثرة، فهي توفر لك محارفًا خاصةً تُساعدك في تحديد مجموعة من الملفات بسرعة كبيرة. هذه المحارف الخاصة يُطلق عليها اسم "المحارف البديلة" (Wildcards). يسمح

استخدام المحارف البديلة (يُعرف أيضًا باسم التعميم) بتحديد بعض الملفات بالاعتماد على نمط مُعيّن موجود في أسمائها. يذكر الجدول الآتي المحارف البديلة ويُبيّن معناها:

الجدول 4-1: المحارف البديلة

المحرف	يطابق
*	أي محرف.
?	أي محرف واحد.
[characters]	أي محرف ينتمي إلى المجموعة characters.
[!characters]	أي محرف لا ينتمي إلى المجموعة characters.
[[:class:]]	أي محرف ينتمي إلى الفئة المعينة.

يعرض الجدول الآتي أكثر فئات الحروف شهرةً:

الجدول 4-2: فئات الحروف الشائعة الاستخدام

الفئة	تطابق
[:alnum:]	أي حرف أبجدي أو رقم.
[:alpha:]	أي حرف أبجدي.
[:digit:]	أي رقم.
[:lower:]	أي حرف أبجدي بحالة الأحرف الصغيرة "abc...".
[:upper:]	أي حرف أبجدي بحالة الأحرف الكبيرة "ABC...".

تجعل المحارف البديلة من الممكن إنشاء أنماط معقدة جدًا لمطابقة أسماء الملفات. يحتوي الجدول الآتي على بعض الأنماط وما الذي تطابقه:

الجدول 4-3: أمثلة عن المحارف البديلة

النمط	يطابق
*	جميع الملفات.
g*	جميع الملفات التي تبدأ بحرف "g".

b*.txt	جميع الملفات التي تبدأ بالحرف "b" وتنتهي باللاحقة ".txt".
Data???	أي ملف يبدأ بالكلمة "Data" ويتبعها ثلاثة محارف.
[abc]*	أي ملف يبدأ بأحد الحروف "a" أو "b" أو "c".
BACKUP.[0-9][0-9][0-9]	أي ملف يبدأ بالعبارة "BACKUP." تتبعها ثلاثة أرقام.
[[:upper:]]*	أي ملف يبدأ بحرف كبير.
[![:digit:]]*	أي ملف لا يبدأ برقم.
*[[:lower:]123]	أي ملف ينتهي بحرف بالحالة الصغيرة أو الأرقام "1" و "2" و "3".

يمكن استخدام المحارف البديلة مع أي أمر يقبل أسماء الملفات كوسيط. سنتحدث بالتفصيل عن المحارف البديلة لاحقًا في الفصل السابع.

مجالات الحروف

إذا كنت قد قدمت من نظام شبيه بِيونكس، أو قرأت أحد الكتب في هذا الموضوع، قد تجد مجموعة الحروف على شكل مجال [A-Z] أو [a-z]. هذه هي مجموعات الحروف التقليدية في أنظمة يونكس وتعمل أيضًا في لينُكس. لكن يجب أن تأخذ حذرك عند استخدامها لأنها قد لا تعطي النتائج المطلوبة إلا إذا كُتِبَت كتابةً صحيحةً. لذا تجنب استخدامها واستخدم فئات الحروف بدلًا منها.

المحارف البديلة تعمل في الواجهة الرسومية أيضًا!

المحارف البديلة مهمة جدًا ليس لأنها تُستخدم بكثرة في سطر الأوامر فحسب، بل وتعمل في الواجهة الرسومية في بعض مدراء الملفات. في نوتاليز (مدير الملفات لسطح مكتب غنوم)، بإمكانك تحديد الملفات من قائمة الخيارات ثم "...Select items Matching". أدخل النمط وستُحدد الملفات المُطابِقَة له. في دولفين أو كُنكرر (مدراء الملفات لسطح مكتب كدي)، تستطيع إدخال المحارف البديلة في شريط المسار (Location bar الموجود أعلى النافذة). على سبيل المثال، إذا أردت عرض جميع الملفات التي تبدأ بحرف "u" في مجلد /usr/bin، اطبع "/usr/bin/u*" في شريط المسار (تأكد من تفعيل خيار تعديل شريط المسار) وستظهر النتائج المُطابَقة.

شَقَّت العديد من أفكار التعامل مع سطر الأوامر طريقها إلى الواجهات الرسومية. هذا أحد الأمثلة الكثيرة التي تجعل الواجهة الرسومية في لينُكس قوية جدًا.

إنشاء المجلدات باستخدام الأمر mkdir

يُستخدم الأمر mkdir لإنشاء المجلدات. شكله العام:

```
mkdir directory...
```

ملاحظة: عندما تتبع ثلاث نقاط أحد الوسائط عند شرح أمرٍ ما، فهذا يعني أن الوسيط يمكن تكراره أكثر من مرّة. إذًا فإن:

```
mkdir dir1
```

يُنشئ مجلد واحد باسم "dir1"، بينما الأمر:

```
mkdir dir1 dir2 dir3
```

يُنشئ ثلاثة مجلدات: "dir1" و "dir2" و "dir3".

نسخ الملفات والمجلدات باستخدام الأمر cp

يُستخدم الأمر cp، الذي ينسخ الملفات أو المجلدات، بطريقتين مختلفتين. الأمر:

```
cp item1 item2
```

يُستخدم لنسخ الملف أو المجلد "item1" إلى المجلد أو الملف "item2". بينما الأمر:

```
cp item... directory
```

يُستخدم لنسخ أكثر من عنصر (سواءً كان ملفًا أم مجلدًا) إلى مجلد معين.

خيارات وأمثلة مفيدة

يحتوي الجدول الآتي على أشهر الخيارات (القصيرة والطويلة) التي تُستخدم مع الأمر cp:

الجدول 4-4: خيارات cp

الخيار	المعنى
`-a, --archive`	نسخ الملفات والمجلدات مع كل خاصياتهم بما فيها الملكية والأذونات. عمومًا، يأخذ النسخ الخاصيات الافتراضية للمستخدم الذي يجري عملية النسخ.

-i, --interactive	طلب موافقة المستخدم قبل استبدال ملف موجود مُسبقًا. سيَستبدل الأمر cp الملف دون أي إشعار إذا لم يُستخدم هذا الخيار.
-r, --recursive	نسخ المجلدات بجميع محتوياتها هذا الخيار (أو الخيار a-) ضروري عند نسخ المجلدات.
-u, --update	نسخ الملفات غير الموجودة أو ذات تاريخ تعديل أحدث من تلك الموجودة في المجلد الهدف عند نسخ الملفات من مجلد إلى آخر.
-v, --verbose	إظهار معلومات عن تقدم عملية النسخ.

أمثلة عن استخدام cp:

الجدول 4-5: أمثلة عن استخدام cp

الأمر	النتيجة
cp file1 file2	إنشاء نسخة من file1 باسم file2. لكن كن حذرًا، فإذا كان الملف file2 موجودًا؛ فسيتم استبداله بالملف file1. أما إذا لم يكن موجودًا فسيُنشأ.
cp -i file1 file2	شبيه بالأمر السابق إلا أنه يطلب تأكيد المستخدم للقيام بعملية النسخ إذا كان الملف file2 موجودًا.
cp file1 file2 dir1	نسخ الملفين file1 و file2 إلى المجلد dir1. لكن يجب أن يكون المجلد dir1 موجودًا من قبل.
cp dir1/* dir2	نسخ جميع محتويات المجلد dir1 إلى المجلد dir2 عن طريق المحارف البديلة. يجب أن يكون المجلد dir2 موجودًا.
cp -r dir1 dir2	نسخ جميع محتويات المجلد dir1 إلى المجلد dir2. إذا لم يكن المجلد dir2 موجودًا فسيُنشأ وسيحتوي على محتويات المجلد dir1 ذاتها. أما إذا كان المجلد dir2 موجودًا فسيُنسخ المجلد dir1 (وجميع محتوياته) إلى المجلد dir2.

نقل وإعادة تسمية الملفات باستخدام mv

ينقل الأمر mv الملفات ويُعيد تسميتها في آنٍ واحد وذلك بالاعتماد على طريقة استخدام هذا الأمر. في كلتا الحالتين، لن يكون هنالك ملف يحمل نفس اسم الملف الأصلي بعد تنفيذ الأمر. يُستخدم الأمر mv استخدامًا يشبه إلى حدٍ ما الأمر cp. الأمر:

```
mv item1 item2
```

ينقل أو يُعيد تسمية الملف أو المجلد "item1" إلى "item2". ويُستخدم الأمر:

```
mv item... directory
```

لنقل ملف أو أكثر من مجلد إلى آخر.

خيارات وأمثلة مفيدة

يُشاطر الأمر mv الأمر cp أغلب خياراته:

الجدول 4-6: خيارات mv

الخيار	المعنى
-i, --interactive	طلب موافقة المستخدم قبل استبدال ملف موجود مُسبقًا. سيستبدل الأمر mv الملف دون أي إشعار إذا لم يُستخدم هذا الخيار.
-u, --update	نقل الملفات غير الموجودة أو ذات تاريخ تعديل أحدث من تلك الموجودة في المجلد الهدف عند نقل الملفات من مجلد إلى آخر.
-v, --verbose	إظهار معلومات عن تقدم عملية النقل.

أمثلة عن استخدام الأمر mv:

الجدول 4-7: أمثلة عن استخدام mv

الأمر	النتيجة
mv file1 file2	نقل الملف file1 إلى file2. إذا كان الملف file2 موجودًا فسيستبدل. إما إذا لم يكن موجودًا فسيُنشأ. في كلتا الحالتين لن يكون الملف file1 موجودًا بعد تنفيذ الأمر.

mv -i file1 file2	شبيه بالأمر السابق إلا أنه يطلب تأكيد المستخدم للقيام بعملية النقل إذا كان الملف file2 موجودًا.
mv file1 file2 dir1	نقل الملفين file1 و file2 إلى المجلد dir1. لكن يجب أن يكون المجلد dir1 موجودًا من قبل.
mv dir1 dir2	إذا لم يكن المجلد dir2 موجودًا فسيُنشأ وستُنقَل محتويات المجلد dir1 إلى dir2 وسيُحذف المجلد dir1. أما إذا كان المجلد dir2 موجودًا فسيُنقل المجلد dir1 (وجميع محتوياته) إلى المجلد dir2.

حذف الملفات والمجلدات باستخدام الأمر rm

يُستخدم الأمر rm لحذف الملفات والمجلدات:

```
rm item...
```

حيث "item" هو ملف أو مجلد أو أكثر.

خيارات وأمثلة مفيدة

يحتوي هذا الجدول على أبرز الخيارات التي تُستخدم مع الأمر rm:

الجدول 4-8: خيارات rm

الخيار	المعنى
-i, --interactive	طلب موافقة المستخدم قبل حذف الملف. سيحذف الأمر rm الملف دون أي إشعار إذا لم يُستخدم هذا الخيار.
-r,--recursive	حذف المجلدات بما تحويه. هذا يعني أنه سيحذف المجلدات الفرعية إذا حوى المجلد عليها. يجب استخدام هذا الخيار إذا أردت حذف مجلد ما.
-f,--force	تجاهل الملفات غير الموجودة. هذا الخيار يبطل تأثير الخيار --interactive.
-v, --verbose	إظهار معلومات عن تقدم عملية الحذف.

أمثلة عن استخدام الأمر rm:

الجدول 4-9: أمثلة عن استخدام rm

الأمر	النتيجة
`rm file1`	حذف الملف file1 دون أي إشعار.
`rm -i file1`	يقوم بنفس عمل الأمر السابق إلا أنه يسأل المستخدم قبل حذف الملف.
`rm -r file1 dir1`	حذف الملف file والمجلد dir1 مع جميع محتوياته.
`rm -rf file1 dir1`	يقوم بنفس عمل الأمر السابق إلا أنه لا يبلغ المستخدم عن عدم وجود file1 أو dir1 ويكمل عملية الحذف بصمت.

كن حذرًا مع الأمر rm!

لا تحتوي الأنظمة الشبيهة بِيونكس كنظام لينُكس على أمر للتراجع عن الحذف. لذا، بمجرد حذفك لأي شيء عن طريق الأمر rm فلن تستطيع استعادته. يفترض نظام لينُكس أنك تدرك ماذا تفعل. كن حذرًا بدرجة أكثر عند التعامل مع المحارف البديلة. تخيل هذا المثال البسيط الذي ستُحذف من خلاله جميع ملفات HTML الموجودة في مجلدٍ ما:

```
rm *.html
```

يقوم الأمر السابق بعمله بدون أيّة مشاكل، لكن لو أضفت فراغًا بغير عمد بين "*" و ".html" كما في الأمر:

```
rm * .html
```

فسيحذف الأمر rm جميع الملفات في المجلد، ومن ثم سيشتكي من عدم وجود ملف باسم ".html".

أنصحك بإتباع الطريقة الآتية: عندما تستخدم المحارف البديلة مع أمر rm (باستثناء التأكد من عدم وجود أخطاء طباعية في الأمر!) فجرّب المحارف البديلة مع الأمر ls. تُمكّنك هذه الطريقة من معرفة الملفات التي سوف تُحذف. اضغط بعد ذلك السهم العلوي لإعادة استدعاء الأمر السابق واستبدل ls بالأمر rm.

إنشاء الوصلات

يُستخدم الأمر ln لإنشاء وصلات صلبة أو رمزية، وذلك بطريقتَين. الأولى:

```
ln file link
```

لإنشاء وصلة صلبة. والثانية:

```
ln -s item link
```

لإنشاء وصلة رمزية حيث "item" عبارة عن ملف أو مجلد.

الوصلات الصلبة

الطريقة الأصلية القديمة الموجودة في نظام يونكس لإنشاء الوصلات هي استخدام الوصلات الصلبة، وذلك عند مقارنتها مع الوصلات الرمزية التي تُعتبر أكثر حداثةً. تملك جميع الملفات افتراضيًا وصلةً صلبةً واحدةً تُعطي الملف اسمه. عندما نُنشئ وصلة صلبة، فإننا نُنشئ قيدًا جديدًا للملف. تعاني الوصلات الصلبة من قصورين مهمَين:

1. لا تستطيع الوصلات الصلبة الإشارة إلى ملف خارج نظام الملفات التي تُنشأ فيه. العبارة السابقة تعني أن الوصلة لا تستطيع الإشارة إلى ملف ليس على القرص نفسه الذي يحتويها.
2. لا يمكن إنشاء وصلة صلبة لمجلد.

على النقيض من الوصلات الرمزية، لا يمكن تمييز الوصلات الصلبة من الملف نفسه. فعندما تعرض محتويات مجلدٍ ما يحتوي على وصلة صلبة فلن ترى أيّة إشارة إلى وجود وصلة صلبة. عندما نحذف وصلة صلبة، فإن محتويات الملف لا تتغير، ويبقى الملف موجودًا ؛ ويستمر وجود الملف إلى أن تحذف جميع الوصلات الصلبة التي تُشير إليه.

من المهم معرفة أنك قد تواجه وصلات صلبة بين الحين والآخر. لكن في الآونة الأخيرة غالبًا ما تُستخدم الوصلات الرمزية، التي سوف نناقشها في الفقرة الآتية.

الوصلات الرمزية

أُنشئت الوصلات الرمزية للتغلب على القصور الموجود في الوصلات الصلبة. تعمل الوصلات الرمزية بإنشاء نوع خاص من الملفات تحتوي نصًا يشير إلى الملف أو المجلد الأصلي. فهي تعمل بآلية تُشبه الآلية الموجودة لإنشاء اختصارات في نظام ويندوز، لكنها تسبق تلك الموجودة في ويندوز بعدّة سنوات ;-) .

من الصعب التفريق ما بين الملف المُشار إليه بواسطة وصلة رمزية، والوصلة الرمزية نفسها. على سبيل المثال، إذا كتبت أيّة بيانات إلى الوصلة الرمزية، فستُكتب أيضًا إلى الملف الأصلي. لكن عند حذف وصلة رمزية، فإن الوصلة وحدها ستُحذف وليس الملف. وإذا حُذِفَ الملف المُشار إليه من قِبل وصلة رمزية، فلن تُحذف هذه الوصلة ولكنها لن تشير إلى أي شيء. تسمى الوصلة في هذه الحالة بالمصطلح: "وصلة مُحَطَّمَة" (broken link). سيُظهِر الأمر ls، في أغلب الحالات، الوصلات المحطمة بلونٍ مختلف كالأحمر لتمييزها.

ربما يكون موضوع الوصلات موضوعًا مربكًا. لكن انتظر قليلًا لأننا سنجرّب كل هذه الأمور. وسُيزال الغموض عنها.

لننشئ مكانًا للتجارب

لما كنّا سنجري عمليات معالجة الملفات، فلنبني مكانًا آمنًا كي نُجرّب فيه. يجب علينا في البداية إنشاء مجلد في المنزل ولنسمه "playground" وذلك -بالطبع- باستخدام أوامر معالجة الملفات التي تعلمناها سابقًا.

إنشاء المجلدات

يُستخدم الأمر mkdir لإنشاء المجلدات. لإنشاء مجلد "playground" علينا أولًا التأكد أننا في مجلد المنزل:

```
[me@linuxbox ~]$ cd
[me@linuxbox ~]$ mkdir playground
```

لكي نجعل بيئة التجارب الخاصة بنا مُسليةً أكثر، فلننشئ مجلدَين اثنَين داخل مجلد "playground" ولنسمهما "dir1" و "dir2". لفعل ذلك، يجب علينا تغيير المجلد الحالي إلى "playground" ومن ثم تنفيذ أمر mkdir آخر:

```
[me@linuxbox ~]$ cd playground
[me@linuxbox playground]$ mkdir dir1 dir2
```

لاحظ أن الأمر mkdir يقبل أكثر من وسيط لإنشاء المجلدَين في أمر واحد.

نسخ الملفات

الخطوة التالية هي الحصول على بعض البيانات ونقلها إلى بيئة التجربة، وذلك بنسخ ملف بالأمر cp، حيث سننسخ الملف passwd من المجلد etc/ إلى مجلد العمل الحالي:

```
[me@linuxbox playground]$ cp /etc/passwd .
```

لاحظ كيف استخدمنا النقطة "." للإشارة بشكل مختصر إلى المجلد الحالي. الآن إذا نفذنا الأمر ls فإننا سنرى الملف الذي نسخناه موجودًا:

```
[me@linuxbox playground]$ ls -l
total 12
drwxrwxr-x 2 me me 4096 2008-01-10 16:40 dir1
```

```
drwxrwxr-x 2 me me 4096 2008-01-10 16:40 dir2
-rw-r--r-- 1 me me 1650 2008-01-10 16:07 passwd
```

الآن، وفقط للتسلية، سنُكرّر عملية النسخ ولكن هذه المرة مع استخدام الخيار"v-" (verbose) كي نرى ماذا يفعل:

```
[me@linuxbox playground]$ cp -v /etc/passwd .
`/etc/passwd' -> `./passwd'
```

قام الأمر cp بعملية النسخ مرّة أُخرى، لكنه أظهر هذه المرّة رسالةً مختصرةً توضح العملية التي يجريها الأمر cp الآن. لاحظ أن الأمر cp يستبدل النسخة الأولى دون أي إشعار. هذا لأن الأمر cp يفترض أنك تعرف ماذا تفعل. سنستخدم الخيار "i-" (interactive) للحصول على إشعار بعملية الاستبدال:

```
[me@linuxbox playground]$ cp -i /etc/passwd .
cp: overwrite `./passwd'?
```

تؤدي الاستجابة إلى المِحث بطباعة الحرف "y" إلى استبدال الملف. بينما تؤدي طباعة أي محرف آخر (على سبيل المثال، "n") إلى ترك الأمر cp الملف وشأنه.

نقل وإعادة تسمية الملفات

لا يبدو الاسم "passwd" مرحًا في بيئة التجارب الخاصة بنا. لذا، فلنغيره إلى اسمٍ آخر!

```
[me@linuxbox playground]$ mv passwd fun
```

ولنكمل الآن تجاربنا بنقل الملف الناتج إلى أحد المجلدات ومن ثم إعادته إلى مكانه الأصلي:

```
[me@linuxbox playground]$ mv fun dir1
```

الأمر السابق لنقل الملف إلى المجلد dir1، الآن الأمر:

```
[me@linuxbox playground]$ mv dir1/fun dir2
```

لنقله من المجلد dir1 إلى المجلد dir2، ومن ثم الأمر:

```
[me@linuxbox playground]$ mv dir2/fun .
```

سيعيده إلى مجلد العمل الحالي. سنجرّب الآن الأمر mv على المجلدات. سننقل أولًا الملف "fun" مُجددًا إلى

المجلد dir1:

```
[me@linuxbox playground]$ mv fun dir1
```

ومن ثم ننقل dir1 إلى dir2 ونتأكد من ذلك بالأمر ls:

```
[me@linuxbox playground]$ mv dir1 dir2
[me@linuxbox playground]$ ls -l dir2
total 4
drwxrwxr-x 2 me me 4096 2008-01-11 06:06 dir1
[me@linuxbox playground]$ ls -l dir2/dir1
total 4
-rw-r--r-- 1 me me 1650 2008-01-10 16:33 fun
```

ولما كان المجلد dir2 موجودًا مسبقًا؛ فنقل الأمر mv المجلد dir1 إلى dir2. إذا لم يكن المجلد dir2 موجودًا، فسيُعيد الأمر mv تسمية المجلد بدل نقله. أخيرًا؛ لنعيد كل شيء إلى مكانه:

```
[me@linuxbox playground]$ mv dir2/dir1 .
[me@linuxbox playground]$ mv dir1/fun .
```

إنشاء الوصلات الصلبة

لنجرب الآن إنشاء الوصلات. سنبدأ بالوصلات الصلبة وسننشئ بعضها كي تُشير إلى الملف كالآتي:

```
[me@linuxbox playground]$ ln fun fun-hard
[me@linuxbox playground]$ ln fun dir1/fun-hard
[me@linuxbox playground]$ ln fun dir2/fun-hard
```

سيكون لدينا الآن ما يُشبه أربع "نسخ" متطابقة تُشير إلى الملف fun (وذلك بربطهم مع رقم العقدة ذاته). لنلقِ نظرة على محتويات المجلد:

```
[me@linuxbox playground]$ ls -l
total 16
drwxrwxr-x 2 me me 4096 2008-01-14 16:17 dir1
drwxrwxr-x 2 me me 4096 2008-01-14 16:17 dir2
-rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun
-rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun-hard
```

ستلاحظ أن الحقل الثاني للملفَين fun و fun-hard يحمل القيمة "4" التي تُمثل عدد الوصلات الصلبة للملف. تذكر أنه سيكون لديك دائمًا وصلة صلبة واحدة للملف على الأقل، لأن اسم الملف يُنشَأ عن طريق وصلة صلبة. إذًا، كيف نستطيع معرفة أن fun و fun-hard يُشيران إلى نفس الملف؟ أمر ls السابق غير مفيد في هذه الحالة. وعلى الرغم من حجمَيّ الملفَين متساويَين (الحقل الخامس) إلا أننا لا نملك طريقة لكي نتأكد من ذلك. يجب علينا أن نتعمق أكثر بموضوع الوصلات لحل هذه المشكلة.

عند التفكير بالوصلات الصلبة، من الجيد تخيل أن الملفات مكونة من جزأين: قسم البيانات الذي يحتوي على محتوى الملف، وقسم "الاسم" الذي يحتوي على اسم الملف. عندما ننشئ وصلة صلبة، فإننا في الواقع ننشئ "نسخة" إضافية تحتوي على اسم الملف فقط، وتشير جميع تلك الوصلات إلى نفس البيانات. يُسنِد النظام جزءًا من القرص الصلب إلى ما يُسمى بالعقدة (inode) التي ترتبط بالقسم الذي يحتوي على الاسم. لذا، كل وصلة صلبة تُشير إلى عقدة تحتوي على محتوى الملف.

يحتوي الأمر ls على آلية لعرض هذه المعلومات وذلك باستخدام الخيار "i-":

```
[me@linuxbox playground]$ ls -li
total 16
12353539 drwxrwxr-x 2 me me 4096 2008-01-14 16:17 dir1
12353540 drwxrwxr-x 2 me me 4096 2008-01-14 16:17 dir2
12353538 -rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun
12353538 -rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun-hard
```

يُمثّل أول الحقول في ناتج الأمر السابق رقم العقدة؛ وكما نلاحظ، لكلا الملفَين fun و fun-hard رقم العقدة ذاته، وهذا ما يؤكد أنهما يُمثلان نفس الملف.

إنشاء وصلات رمزية

أُنشئت الوصلات الرمزية لتتغلب على القصور الموجود في الوصلات الصلبة: الوصلات الصلبة لا تستطيع الإشارة إلى ملف خارج حدود القرص الذي يحتويها. بالإضافة إلى عدم مقدرة الوصلات الصلبة على الإشارة إلى المجلدات. الوصلات الرمزية هي ملفات من نوع خاص تحتوي على رابط نصي للملف أو المجلد الهدف.

إنشاء الوصلات الرمزية شبيه بإنشاء الوصلات الصلبة:

```
[me@linuxbox playground]$ ln -s fun fun-sym
[me@linuxbox playground]$ ln -s ../fun dir1/fun-sym
[me@linuxbox playground]$ ln -s ../fun dir2/fun-sym
```

المثال الأول بسيط للغاية، بكل بساطة، أضفنا الخيار "s-" لإنشاء وصلة رمزية بدلًا من وصلة صلبة. لكن ماذا

عن الأمرَين التاليَين؟ تذكر أننا عندما ننشئ وصلةً رمزيةً فإننا نُنشئ ملفًا يحتوي على وصف نصي للمسار النسبي للملف الهدف. لنلق نظرة على ناتج الأمر ls لإزالة ما تبقى من غموض:

```
[me@linuxbox playground]$ ls -l dir1
total 4
-rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun-hard
lrwxrwxrwx 1 me me    6 2008-01-15 15:17 fun-sym -> ../fun
```

السطر الخاص بالقيد "fun-sym" يُظهر أنه وصلة رمزية، وذلك بوجود الحرف "l" في الحقل الأول، وتشير هذه الوصلة إلى الملف "../fun" وهذا صحيح لأن الملف "fun" يقع في المجلد الذي يعلو المجلد الذي يحتوي على الوصلة الرمزية. لاحظ أيضًا أن حجم الوصلة هو 6، الذي يُمثل عدد الأحرف في العبارة "../fun" عوضًا عن حجم الملف الأصلي الذي تشير الوصلة إليه.

بإمكانك تحديد المسار المطلق للملف عند إنشاء الوصلات الرمزية:

```
ln -s /home/me/playground/fun dir1/fun-sym
```

أو المسارات النسبية كما فعلنا في المثال السابق. من الأفضل استخدام المسارات النسبية لأنها تسمح بنقل المجلد الحاوي على الوصلة الرمزية أو إعادة تسميته دون تحطيمها.

بالإضافة إلى الملفات، فيمكن بكل سهولة أن تُشير الوصلات الرمزية إلى المجلدات:

```
[me@linuxbox playground]$ ln -s dir1 dir1-sym
[me@linuxbox playground]$ ls -l
total 16
drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir1
lrwxrwxrwx 1 me me    4 2008-01-16 14:45 dir1-sym -> dir1
drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir2
-rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun
-rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun-hard
lrwxrwxrwx 1 me me    3 2008-01-15 15:15 fun-sym -> fun
```

حذف الملفات والمجلدات

كما ناقشنا سابقًا، يُستخدم الأمر rm لحذف الملفات والمجلدات. سنستخدمه لكي "ننظف" الفوضى التي أنشأناها. لنحذف في البداية إحدى الوصلات الصلبة:

```
[me@linuxbox playground]$ rm fun-hard
[me@linuxbox playground]$ ls -l
total 12
drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir1
lrwxrwxrwx 1 me me    4 2008-01-16 14:45 dir1-sym -> dir1
drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir2
-rw-r--r-- 3 me me 1650 2008-01-10 16:33 fun
lrwxrwxrwx 1 me me    3 2008-01-15 15:15 fun-sym -> fun
```

وكما كان مُتوقعًا، قد حُذِف الملف fun-hard ونقص عدد الوصلات للملف fun من أربع وصلات إلى ثلاث، وذلك في الحقل الثاني من ناتج الأمر ls. سنحذف الآن الملف fun مُستخدمين الخيار "-i" لكي نرى ماذا سيحدث:

```
[me@linuxbox playground]$ rm -i fun
rm: remove regular file `fun'?
```

اطبع "y" في المِحث وسيُحذف الملف. لنلق نظرة على مخرجات الأمر ls؛ لاحظ ماذا حصل إلى fun-sym. لما كانت الوصلة الرمزية تُشير حاليًا إلى ملف غير موجود، فإن الوصلة قد تحطمت:

```
[me@linuxbox playground]$ ls -l
total 8
drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir1
lrwxrwxrwx 1 me me    4 2008-01-16 14:45 dir1-sym -> dir1
drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir2
lrwxrwxrwx 1 me me    3 2008-01-15 15:15 fun-sym -> fun
```

تُعِدّ أغلب توزيعات لينُكس الأمر ls لكي يُظهر الوصلات المحطمة ويُميزها. في توزيعة أوبنتو، تظهر الوصلات المحطمة بلون خلفية أحمر. وجود الوصلات المحطمة ليس خطرًا بحد ذاته، لكنه قد يؤدي إلى بعض الفوضى. سنشاهد رسالة شبيهة بالرسالة الآتية إذا حاولنا استخدام وصلة محطمة:

```
[me@linuxbox playground]$ less fun-sym
fun-sym: No such file or directory
```

لنحذف الآن الوصلات الرمزية:

```
[me@linuxbox playground]$ rm fun-sym dir1-sym
```

```
[me@linuxbox playground]$ ls -l
total 8
drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir1
drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir2
```

أحد أهم الأشياء التي يجب تذكرها عند التعامل مع الوصلات الرمزية هي أن معظم العمليات تُنفذ على الملف الأصلي وليست على الوصلة. الأمر rm هو استثناء. فعندما تُحذف وصلة رمزية، فستُحذف الوصلة فقط وليس الملف الأصلي.

في النهاية، سنحذف مجلد "playground". وذلك بالعودة إلى مجلد المنزل واستخدام الأمر rm مع الخيار "-r" لحذف المجلد مع كافة محتوياته (بما فيها المجلدات الفرعية):

```
[me@linuxbox playground]$ cd
[me@linuxbox ~]$ rm -r playground
```

إنشاء الوصلات الرمزية في الواجهة الرسومية

يوفّر مدراء الملفات في غنوم وكدي طريقةً سهلةً وتلقائيةً لإنشاء الوصلات الرمزية. في غنوم، اضغط على Ctrl-shift عند سحب وإفلات الملفات لإنشاء وصلات رمزية بدلًا من نسخها (أو نقلها). أما في كدي، فسوف تظهر قائمة صغيرة عند إفلات الملف، تحتوي على خيارات لنسخ أو نقل أو إنشاء وصلة للملف.

الخلاصة

لقد شرحنا في هذا الفصل أشياءً كثيرة. أعد إنشاء "بيئة التجارب" مرارًا وتكرارًا حتى تألف استخدام الأوامر. من المهم جدًا فهم الأوامر الأساسية المُستخدمة في تعديل الملفات بالإضافة إلى المحارف البديلة. خذ وقتك في توسيع بيئة التجارب بإضافة عدد من الملفات والمجلدات، استخدم المحارف البديلة لتحديد أكثر من ملف لإجراء العمليات المختلفة عليها. موضوع الوصلات يبدو للوهلة الأولى مربكًا. لذا، خذ وقتك لتعلم آلية عملها.

الفصل الخامس:

التعامل مع الأوامر

لقد تعاملنا، إلى هذه النقطة في الكتاب، مع سلسلةٍ من الأوامر الغامضة ذات الخيارات الغريبة! سنحاول في هذا الفصل إزالة جزء من الغموض الذي يُحيط بتلك الأوامر. وسنتعلّم أيضًا آلية إنشاء أوامر خاصة بنا. الأوامر التي ستُناقش في هذا الفصل هي:

- `type` – تحديد طريقة تفسير اسم الأمر.
- `which` – عرض مسار الملف التنفيذي المرتبط مع الأمر.
- `help` – الحصول على المساعدة للأوامر المُضَمّنَة في الصدفة.
- `man` – عرض صفحة الدليل.
- `apropos` – عرض قائمة بالأوامر الملائمة.
- `info` – عرض القيد الخاص بالأمر في صفحات `info`.
- `whatis` – عرض شرح مختصر عن الأمر.
- `alias` – إنشاء أمر بديل لأمر آخر.

ما هي الأوامر؟

يجب أن ينتمي أي أمر إلى إحدى المجموعات الأربع الآتية:

1. **ملف تنفيذي** كالملفات التي شاهدناها في المجلد `/usr/bin`. تحتوي هذه المجموعة على الملفات الثنائية التي بُنيت من برمجيات مكتوبة بلغة `C/C++` أو البرمجيات المكتوبة بلغات النصوص البرمجية أو السكربتات (scripting languages) كلغة بيرل وبيثون وروبي... إلخ.
2. **أوامر مُضمّنة في الصدفة نفسها.** تدعم صدفة bash عددًا من الأوامر المُضمّنة فيها تسمى "shell builtins". الأمر cd، على سبيل المثال، هو أمر مُضمّن في الصدفة.
3. **دوال الشِل.** دوال الشِل هي سكربتات صغيرة مدمجة في "البيئة". سنناقش ضبط البيئة وكتابة دوال الشِل في فصول لاحقة. كل ما يلزمنا معرفته الآن هو وجود هذه الدوال.
4. **الأوامر البديلة.** الأوامر التي نُعرّفها بأنفسنا. وتبنى عادةً من باقي الأوامر.

تعيين نوع الأمر

من المفيد معرفة إلى أيّة مجموعة من المجموعات الأربعة السابقة ينتمي أمرٌ ما. يوفر نظام لينُكس طريقتَين لتحديد ذلك.

عرض نوع الأمر باستخدام type

الأمر type هو أمر مُضمّن بالصدفة يُحدِد نوع الأمر الذي يُمرَر إليه. شكل الأمر العام:

```
type command
```

حيث "command" هو اسم الأمر الذي نريد الاستعلام عن نوعه. بعض الأمثلة عن استخدام هذا الأمر:

```
[me@linuxbox ~]$ type type
type is a shell builtin
[me@linuxbox ~]$ type ls
ls is aliased to `ls --color=tty'
[me@linuxbox ~]$ type cp
cp is /bin/cp
```

شاهدنا في المثال السابق نتائج ثلاثة أوامر مختلفة. لاحظ أن الأمر ls (أُخذت النتيجة من توزيعة فيدورا) ما هو إلا "أمر بديل" للأمر ls لكن مع إضافة الخيار "--color=tty". أصبحنا الآن نعرف لماذا يعرض الأمر ls نتائجه بالألوان!

عرض مسار الملف التنفيذي باستخدام which

يوجد في بعض الأحيان أكثر من نسخة للملف التنفيذي المُثبّت على نظامك. ربما يكون ذلك الوضع غير شائع في أنظمة سطح المكتب، إلا أنه شائع جدًا في الخوادم. نستخدم الأمر which لتحديد مسار الملف التنفيذي لأمر ما:

```
[me@linuxbox ~]$ which ls
/bin/ls
```

لا يعمل الأمر which إلا مع الملفات التنفيذية، ولا يعمل مع الأوامر المُضمنة أو الأوامر البديلة التي هي بديل عن الملفات التنفيذية. فعند تجربة عرض معلومات أمر مدمج بالصدفة وليكن cd، فإنك ستحصل على رسالة الخطأ الآتية:

```
[me@linuxbox ~]$ which cd
/usr/bin/which: no cd in (/opt/jre1.6.0_03/bin:/usr/lib/qt-
3.3/bin:/usr/kerberos/bin:/opt/jre1.6.0_03/bin:/usr/lib/ccache:/usr/l
ocal/bin:/usr/bin:/bin:/home/me/bin)
```

الرسالة السابقة هي طريقة مُنمّقة لقول: "command not found".

الحصول على التوثيق للأوامر

نستطيع الآن بعد معرفة نوع الأمر البحث عن التوثيق المتوفر له.

الحصول على المساعدة للأوامر المُضمّنة في الصدفة

تحتوي صدفة bash على خاصية مُضمّنة تُستخدم للحصول على المساعدة للأوامر المُضمّنة فيها. اكتب الكلمة "help" يتبعها اسم الأمر المضمن في الصدفة؛ على سبيل المثال:

```
[me@linuxbox ~]$ help cd
cd: cd [-L|[-P [-e]]] [dir]
Change the shell working directory.
Change the current directory to DIR. The default DIR is the value of
the HOME shell variable.

The variable CDPATH defines the search path for the directory
containing DIR. Alternative directory names in CDPATH are separated
by a colon (:). A null directory name is the same as the current
directory. If DIR begins with a slash (/), then CDPATH is not used.
If the directory is not found, and the shell option `cdable_vars' is
set, the word is assumed to be a variable name. If that variable
has a value, its value is used for DIR.

Options:
-L    force symbolic links to be followed
-P    use the physical directory structure without following symbolic
links
-e    if the -P option is supplied, and the current working directory
cannot be determined successfully, exit with a non-zero status
```

```
The default is to follow symbolic links, as if `-L' were specified.

Exit Status:
Returns 0 if the directory is changed, and if $PWD is set
successfully when -P is used; non-zero otherwise.
```

ملاحظة: عند ورود الأقواس المربعة "[]" في شرح خيارات أي أمر فهذا يعني أنه اختياري. وعند ورود خط عمودي "|" بين خيارين فهذا يعني أنه بإمكانك استخدام أحد الخيارين فقط. في الحالة السابقة (cd):

```
cd: cd [-L|[-P [-e]]] [dir]
```

هذا يعني أن الأمر cd يمكن أن يقبل اختياريًّا أحد الخيارين "L-" أو "P-" ويمكن أيضًا استخدام الخيار e- بعد الخيار P-. ومن ثم يتبعه (اختياريًّا أيضًا) الوسيط "dir".

وعلى الرغم من أن ناتج help للأمر cd دقيق جدًا وكامل، إلا أنك لا تستطيع الاعتماد عليه كطريقة للتعلم (دروس)! وكما لاحظت فإنه يحتوي العديد من الأمور التي لم نتكلم عنها بعد! لا تقلق، فسنشرح تلك المواضيع قريبًا.

عرض معلومات الاستخدام باستخدام الخيار help--

تدعم العديد من البرمجيات التنفيذية الخيار "help--" الذي يعرض شرح للشكل العام للأمر والخيارات التي يقبلها. على سبيل المثال:

```
[me@linuxbox ~]$ mkdir --help
Usage: mkdir [OPTION] DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

     -Z,   --context=CONTEXT (SELinux) set security context to CONTEXT
Mandatory arguments to long options are mandatory for short options
too.
     -m,   --mode=MODE set file mode (as in chmod), not a=rwx – umask
     -p,   --parents no error if existing, make parent directories as
needed
     -v,   --verbose print a message for each created directory
           --help display this help and exit
           --version output version information and exit
Report bugs to <bug-coreutils@gnu.org>.
```

لا تدعم بعض البرمجيات الخيار "--help". لكن جربه على أي حال، لأن الأمر غالبًا ما سيُظهِر رسالة خطأ وبعض المعلومات عن طريقة الاستخدام الخاصة به.

عرض صفحات الدليل man

توفر أغلب البرمجيات التنفيذية التي أُنشئت للاستخدام مع سطر الأوامر توثيقًا رسميًا يُسمى صفحات الدليل (manual أو man). يوجد برنامج يُستخدم لعرض تلك الصفحات يُسمى man. ويُستخدم كالآتي:

```
man program
```

حيث "program" هو اسم الأمر الذي نريد مشاهدة صفحات الدليل الخاصة به.

تختلف صفحات man في التنسيق إلا أنها تحتوي عادةً العنوان وملخص عن الشكل العام للأمر وشرح عن الغرض منه، ومن ثم قائمة تشرح جميع الخيارات التي يقبلها الأمر. لكن صفحات man لا تحتوي على أمثلة والغرض منها هو أن تكون مرجعًا وليست درسًا. على سبيل المثال، إذا أردنا مشاهدة صفحة الدليل للأمر ls:

```
[me@linuxbox ~]$ man ls
```

يَستخدم الأمر man، في أغلب توزيعات لينُكس، برمجية less لعرض صفحات الدليل. لذا، فإن جميع الأوامر المألوفة التي استخدمتها من قبل مع less تعمل مع man.

"الدليل"، الذي يعرض الأمر man الصفحات الموجودة بداخله، مُقسّم إلى أقسام لا تشرح الأوامر التي يستخدمها المستخدم العادي فحسب، بل تتعداها إلى شرح الأوامر الخاصة بمدير النظام والواجهات البرمجية وبنية ملفات الإعدادات المختلفة والكثير. يُبيّن الجدول الآتي بنية هذا الدليل:

الجدول 5-1: تنظيم صفحات الدليل man

القسم	المحتوى
1	أوامر المستخدم.
2	الواجهة البرمجية للنواة.
3	الواجهة البرمجية لمكتبة لغة C.
4	الملفات الخاصة كالأجهزة والأقراص.
5	صيغ الملفات.
6	الألعاب والترفيه، كالشاشات المؤقتة.

7	متفرقات.
8	أوامر إدارة الأنظمة.

يلزمنا في بعض الأحيان البحث في قسم خاص من الدليل كي نحصل على ما نريد. قد تحدث هذه الحالة عندما نريد البحث عن بُنية ملف ونحصل على شرح للأمر بدلًا منها. سنحصل دائمًا على أول مطابقة لكلمة البحث إذا لم نُحدد رقم القسم (غالبًا ما سيكون القسم الأول). نستخدم الأمر man على النحو الآتي إذا أردنا تحديد رقم القسم:

```
man section search_term
```

على سبيل المثال، الأمر:

```
[me@linuxbox ~]$ man 5 passwd
```

سيعرض صفحة الدليل لبُنية الملف /etc/passwd.

apropos: عرض الأوامر الملائمة

يمكن أيضًا البحث في صفحات الدليل عن مطابقة مبنية على عبارة البحث. قد يبدو الأمر بسيطًا لكنه مفيد في بعض الأحيان. سيعرض المثال الآتي نتائج البحث في صفحات الدليل لعبارة البحث "floppy":

```
[me@linuxbox ~]$ apropos floppy
create_floppy_devices (8) - udev callout to create all possible
                            floppy device based on the CMOS type
fdformat              (8) - Low-level formats a floppy disk
floppy                (8) - format floppy disks
gfloppy               (1) - a simple floppy formatter for the GNOME
mbadblocks            (1) - tests a floppy disk, and marks the bad
                             blocks in the FAT
mformat               (1) - add an MSDOS filesystem to a low-level
                            formatted floppy disk
```

الحقل الأول من الناتج السابق هو اسم صفحة الدليل، أما الحقل الثاني فيُظهر رقم القسم. لاحظ أن استخدام الأمر man مع الخيار "-k" يقوم بنفس عمل الأمر apropos.

عرض شرح مختصر عن أحد الأوامر باستخدام whatis

يعرض الأمر whatis اسم وسطر واحد من صفحة الدليل للكلمة التي طُوبِقت.

```
[me@linuxbox ~]$ whatis ls
ls              (1) - list directory contents
```

أقسى صفحات الدليل على الإطلاق!

كما رأيت، إن الغرض من صفحات الدليل المتوفرة مع لينُكس وباقي الأنظمة الشبيهة بِيونكس هو تقديم مرجع شامل وليس مجموعة دروس عن الأوامر. من الصعب قراءة العديد من صفحات الدليل، لكنني أظن أن الجائزة الكبرى للصعوبة تذهب إلى صفحة bash في الدليل. ألقيتُ نظرةً على تلك الصفحة عند إعدادي للكتاب كي أتأكد أنني قد شرحتُ أغلب مواضيعها. عندما تُطبَع تلك الصفحة، فإنها ستأخذ حوالى 80 صفحة من الورق ذات المحتوى الكثيف جدًا. بالإضافة إلى ذلك، تكون بنية الصفحة غير منطقية أبدًا للمستخدم الجديد.

لكن لننظر إلى الجانب المشرق، فإن تلك الصفحة دقيقة وكاملة للغاية! لذا، اطلع عليها إذا كنت تجرؤ! وتتطلّع إلى اليوم الذي تكون فيه جميع محتوياتها منطقية.

عرض قيد info الخاص ببرنامج

وفّر مشروع غنو بديلًا عن صفحات man لبرمجياتهم أسموها "صفحات info". تُعرَض هذه الصفحات ببرنامج يحمل الاسم "info". تحتوي صفحات info على روابط فائقة كتلك الموجودة في صفحات الوِب. مثال:

```
File: coreutils.info, Node: ls invocation, Next: dir invocation,
Up: Directory listing



10.1 `ls': List directory contents
==================================



The `ls' program lists information about files (of any type,
including directories). Options and file arguments can be intermixed
```

```
arbitrarily, as usual.


   For non-option command-line arguments that are directories, by
default `ls' lists the contents of directories, not recursively, and
omitting files with names beginning with `.'. For other non-option
arguments, by default `ls' lists just the filename. If no non-option
argument is specified, `ls' operates on the current directory, acting
as if it had been invoked with a single argument of `.'.
By default, the output is sorted alphabetically, according to the
  --zz-Info: (coreutils.info.gz)ls invocation, 63 lines --Top----------
```

يقرأ برنامج info صفحات info التي تكون بنيتها بُنية شجرية مُقسّمة إلى عقد. كل عقدة تحتوي على موضوع واحد. تحتوي صفحات info على روابط فائقة التي تُمكنك من الانتقال من عقدة إلى أُخرى. يُفعّل الرابط الفائق (الذي نستطيع تمييزه برمز النجمة الذي يسبقه) عند تحريك المؤشر إليه والضغط على زر Enter.

لاستخدام صفحات info، اطبع الكلمة "info" يتبعها (اختياريًّا) اسم البرنامج. يعرض الجدول الآتي الأوامر التي يمكن استخدامها مع صفحات info:

الجدول 5-2: أوامر info

الأمر	النتيجة
?	عرض صفحة المساعدة.
PgUp أو Backspace	عرض الصفحة السابقة.
PgDn أو Space	عرض الصفحة التالية.
N	عرض العقدة التالية.
P	عرض العقدة السابقة.
U	عرض العقدة الأب للعقدة الحالية، غالبًا ما تكون عبارة عن قائمة التنقل.
Enter	اتّباع الرابط الفائق الموجود عند المؤشر.
q	إنهاء البرنامج.

أغلب الأوامر التي ناقشناها حتى الآن هي جزء من حزمة "coreutils" التابعة لمشروع غنو.

ملفات README وباقي ملفات التوثيق

تُخزِّن العديد من حزم البرمجيات المُثبّتة على جهازك ملفات التوثيق في مجلد usr/share/doc/. أغلب تلك الملفات مُخزَّنة على شكل ملفات نصية بسيطة. بعضها الآخر مخزن بصيغة HTML التي يمكن أن تُعرَض في متصفحات الوِب. يمكن أن نواجه بعض الملفات بامتداد ".gz". هذا يعني أن هذه الملفات مضغوطة باستخدام تقنية gzip. تحتوي حزمة gzip على نسخة خاصة من less تُدعى zless، تستطيع عرض محتويات الملفات النصية المضغوطة بتقنية gzip.

إنشاء أومرك الخاصة باستخدام alias

هذه هي أولى خطواتنا في طريق البرمجة! سنُنشئ أوامرنا الخاصة باستخدام alias. لكن قبل أن نبدأ، نحتاج إلى تعلم خدعة بسيطة في سطر الأوامر؛ من الممكن وضع أكثر من أمر في سطر واحد وذلك بالفصل فيما بينهما بفاصلة منقوطة ";"، كالآتي:

```
command1; command2; command3...
```

وهذا مثال نستخدم فيه الخدعة البسيطة السابقة:

```
[me@linuxbox ~]$ cd /usr; ls; cd -
bin   games    kerberos  lib64     local  share  tmp
etc   include  lib       libexec   sbin   src
/home/me
[me@linuxbox ~]$
```

كما لاحظت، دمجنا ثلاثة أوامر في سطر واحد. غيّرنا، في البداية، المجلد الحالي إلى usr/ ومن ثم عرضنا ملفاته، وفي النهاية عدنا إلى المجلد السابق (بالأمر "- cd"). لنحول الآن سلسلة الأوامر السابقة إلى أمر جديد باستخدام alias. إن ما يجب علينا فعله هو التفكير باسم للأمر الجديد. لنجرب الكلمة "test". لكن قبل أن نفعل ذلك، يجب علينا التأكد من أن الاسم "test" غير مستخدم من قبل؛ وسنستخدم لهذا الغرض الأمر type:

```
[me@linuxbox ~]$ type test
test is a shell builtin
```

للأسف، الاسم "test" محجوز مسبقًا. لنجرب "foo":

```
[me@linuxbox ~]$ type foo
bash: type: foo: not found
```

ممتاز، الاسم "foo" غير مستخدم. لذا، لننشئ الأمر الخاص بنا:

```
[me@linuxbox ~]$ alias foo='cd /usr; ls; cd -'
```

لاحظ أن بنية الأمر alias هي كالآتي:

```
alias name='string'
```

بعد اسم الأمر "alias" فإننا نعطي الأمر البديل اسمه متبوعًا (بدون أيّة فراغات) بإشارة المساواة ومن بعدها سلسلة نصية تحتوي على الأمر مُحاطًا بعلامتَيّ اقتباس. نستطيع استخدام الأمر الخاص بنا بعد تعريفه في أي مكان في الصدفة (يمكن استخدام أحد الأوامر فيه). لنُجرّب ذلك:

```
[me@linuxbox ~]$ foo
bin   games    kerberos  lib64     local  share  tmp
etc   include  lib       libexec   sbin   src
/home/me
[me@linuxbox ~]$
```

ولنجرب أيضًا الأمر type على الأمر البديل الخاص بنا:

```
[me@linuxbox ~]$ type foo
foo is aliased to `cd /usr; ls ; cd -'
```

لإزالة الأمر البديل، نستخدم الأمر unalias. كما في المثال الآتي:

```
[me@linuxbox ~]$ unalias foo
[me@linuxbox ~]$ type foo
bash: type: foo: not found
```

وعلى الرغم من أننا تجنبنا عن قصد تسمية الأمر البديل باسم موجود مسبقًا، لكن ذلك الاستخدام شائع. نفعل ذلك لتضمين خيارات تُستخدم بكثرة مع بعض الأوامر الشهيرة. على سبيل المثال، شاهدنا مُسبقًا أن الأمر ls ما هو إلا أمرٌ بديل لإضافة خيار إظهار النتائج بالألوان:

```
[me@linuxbox ~]$ type ls
```

```
ls is aliased to `ls --color=tty'
```

لمعرفة جميع الأوامر البديلة المُعرّفة في البيئة لدينا، نستخدم الأمر alias بدون أيّة وسائط. هذه بعض الأوامر البديلة الموجودة في توزيعة فيدورا، جربها أو حاول معرفة معناها:

```
[me@linuxbox ~]$ alias
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
```

هناك مشكلة صغيرة جدًا مع تعريف الأوامر البديلة في سطر الأوامر؛ ستختفي الأوامر البديلة عند إنهاء جلسة الصدفة. سنتعرف في فصولٍ قادمة على آلية إضافة الأوامر البديلة إلى ملفات البيئة كي تُعرّف في كل مرّة نسجل دخولنا فيها. لكن الآن استمتع بأنك قد تعلمت أول دروسك البرمجية وخطيت خطوة إلى الأمام في عالم برمجة الشِل!

الخلاصة

الآن بعد أن تعلمت طريقة البحث عن التوثيق للأوامر، أصبحتَ تستطيع إيجاد التوثيق لجميع الأوامر التي تعلمناها إلى الآن. ادرس الخيارات الإضافية الموجودة لكل أمر وجربها!

الفصل السادس:

إعادة التوجيه

سنستكشف في هذا الفصل إحدى أكثر ميزات سطر الأوامر إثارةً ومتعةً، ألا وهي إعادة توجيه الدخل والخرج "I/O redirection" ("I/O " هي اختصار للعبارة "input/output" أي الدخل والخرج). نستطيع باستخدام هذه الميزة، إعادة توجيه مجرِيَيّ الدخل والخرج من وإلى الملفات، بالإضافة إلى ربط أكثر من أمر باستخدام "الأنابيب". سنتعرف على الأوامر الآتية لاستكشاف هذه الميزة:

- `cat` – لمّ (Concatenate) الملفات.
- `sort` – ترتيب الأسطر النصية.
- `uniq` – التبليغ عن أو حذف الأسطر المكررة.
- `grep` – عرض الأسطر التي تُطابِق نمطًا محددًا.
- `wc` – عرض عدد الأسطر والكلمات وعدد البايتات في ملف.
- `head` – عرض القسم الأول من الملف (السطور الأولى).
- `tail` – عرض القسم الأخير من الملف (السطور الأخيرة).
- `tee` – القراءة من مجرى الدخل القياسي والكتابة إلى مجرى الخرج القياسي وإلى الملفات.

مجاري الدخل والخرج والخطأ القياسية

العديد من البرامج التي تعاملنا معها إلى الآن تطبع مخرجات من نوع ما. تنقسم هذه المخرجات إلى نوعين: النوع الأول هو ناتج تنفيذ البرنامج، أي البيانات التي يُفترض على البرنامج أن يطبعها. أما النوع الثاني فهو رسائل الخطأ التي تخبرنا ماهية المشكلة التي تمنع البرنامج من تنفيذ مهمته. إذا نظرنا إلى الأمر `ls`، فإننا سنرى أنه يطبع النتائج ورسائل الخطأ إلى الشاشة.

ولمجاراة السِمة الخاصة بِيونكس: "كل شيء ملف"؛ تُرسِل البرامج (كالأمر `ls`) مخرجاتها إلى ملف خاص يُسمى "مجرى الخرج القياسي" (يُشار إليه عادةً بالكلمة `stdout`). ورسائل الخطأ إلى ملف آخر يُسمى "مجرى الخطأ القياسي" (`stderr`). يرتبط كلا المجريَين افتراضيًا بالشاشة، ولا يُحفظ إلى الملفات العادية.

بالإضافة إلى ذلك، تقبل العديد من البرامج الإدخال من ما يُسمى "مجرى الدخل القياسي" (`stdin`) الذي

يكون -افتراضيًّا- مرتبطًا بلوحة المفاتيح.

تسمح آلية إعادة توجيه الدخل والخرج بتغيير المكان الذي سيذهب إليه الخرج والمكان الذي سيأتي منه الدخل. عمومًا، يذهب الخرج إلى الشاشة ويأتي الدخل من لوحة المفاتيح؛ لكننا نستطيع مع آلية إعادة توجيه الدخل والخرج تغيير ذلك.

إعادة توجيه مجرى الخرج القياسي

تسمح لنا آلية إعادة توجيه الدخل والخرج بتحديد إلى أين ستذهب مخرجات البرامج. لإعادة توجيه مجرى الخرج القياسي إلى ملف آخر عدا الشاشة؛ نستخدم معامل إعادة التوجيه الذي يُرمز له بالرمز ">" ويتبعه مسار الملف المُراد إعادة التوجيه إليه. لكن لماذا نريد فعل ذلك؟ عادةً تكون هنالك فائدة من حفظ مخرجات أمرٍ ما في ملف. على سبيل المثال، بإمكاننا إخبار الصدفة أن تُرسِل مخرجات الأمر `ls` إلى الملف `ls-output.txt` بدلًا من الشاشة:

```
[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt
```

لقد أنشأنا قائمةً طويلةً بمحتويات المجلد `/usr/bin` وأرسلناها إلى الملف `ls-output.txt`. لنرَ المخرجات التي أُعيد توجيهها من الأمر السابق:

```
[me@linuxbox ~]$ ls -l ls-output.txt
-rw-rw-r-- 1 me me 167878 2008-02-01 15:07 ls-output.txt
```

إذا حاولنا عرض محتويات الملف `ls-output.txt` باستخدام الأمر `less`، فسنشاهد أنه يحتوي بالفعل على ناتج الأمر `ls`:

```
[me@linuxbox ~]$ less ls-output.txt
```

الآن، لنعيد اختبار إعادة التوجيه السابق، لكن هذه المرة سنعبث في الأمر قليلًا. سنغيّر مسار المجلد إلى مسار غير موجود:

```
[me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt
ls: cannot access /bin/usr: No such file or directory
```

لقد تلقينا رسالة خطأ. وهذا شيء منطقي لأن المسار `/bin/usr` غير موجود. لكن لماذا أُظهرت رسالة الخطأ على الشاشة بدلًا من إعادة توجيهها إلى الملف `ls-output.txt`؟ السبب هو أن الأمر `ls` لا يُرسِل رسائل الخطأ إلى مجرى الخرج النظامي، وإنما يرسلها (كباقي برامج يونكس) إلى مجرى الخطأ القياسي. ولأننا أعدنا

توجيه مجرى الخرج القياسي فقط وليس مجرى الخطأ، فما تزال رسائل الخطأ تُرسَل إلى الشاشة. سنتعلم بعد دقيقة واحدة كيف نحول مجرى الخطأ القياسي، لكن لنلق نظرة أولًا على ما حصل لملف الخرج:

```
[me@linuxbox ~]$ ls -l ls-output.txt
-rw-rw-r-- 1 me me 0 2008-02-01 15:08 ls-output.txt
```

الحجم التخزيني للملف هو صفر! السبب هو أننا استخدمنا المعامل ">" الذي يُعيد دائمًا الكتابة على الملف. ولأن الأمر ls لم يُخرِج أية مخرجات سوى رسالة الخطأ، فإن معامل إعادة التوجيه قد بدأ بإعادة الكتابة فوق الملف ومن ثم توقف بسبب الخطأ، مما أدى إلى حذف جميع محتويات الملفات. في الواقع، إذا أردنا حذف جميع محتويات ملفٍ ما (أو إنشاء ملف فارغ جديد)، نستطيع استخدام الخدعة البسيطة الآتية:

```
[me@linuxbox ~]$ > ls-output.txt
```

بكل بساطة، يؤدي استخدام معامل إعادة التوجيه بدون أي أمر إلى حذف محتويات الملف أو إنشاء ملف فارغ جديد.

إذًا، كيف نستطيع أن نلحق مخرجات جديدة إلى ملف موجود مسبقًا عوضًا عن إعادة كتابته كل مرة؟ نستخدم المعامل ">>" لهذا الغرض كالآتي:

```
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
```

سنلحق المخرجات إلى الملف باستخدام المعامل ">>". إذا لم يكن الملف موجودًا، فسيُنشَأ كما في المعامل ">"، لنجرّب ذلك:

```
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
[me@linuxbox ~]$ ls -l ls-output.txt
-rw-rw-r-- 1 me me 503634 2008-02-01 15:45 ls-output.txt
```

كررنا الأمر ثلاث مرات مما أدى إلى جعل الحجم التخزيني للملف أكبر بثلاث مرات.

إعادة توجيه مجرى الخطأ القياسي

لا يتوفر لمجرى الخطأ القياسي مُعامل توجيه خاص به. سنحتاج إلى الإشارة إلى "مقبض الملف" لإعادة توجيه مجرى الخطأ. يمكن لتطبيق ما أن يُرسِل المخرجات إلى واحد من عدة مجاري ملفات مرقمة. وعلى الرغم من أننا أشرنا إلى أول ثلاثة منها بمجرى الدخل والخرج والخطأ، إلا أن الصدفة تعتبرهم (داخليًا) مقابض

الملفات صفر وواحد واثنان على الترتيب، توفر الصدفة آلية لإعادة توجيه الملفات باستخدام أرقام المقابض. ولأن مجرى الخطأ القياسي يُقابله مقبض الملف 2؛ فيُمكننا كتابة الأمر الآتي لإعادة توجيه مجرى الخطأ:

```
[me@linuxbox ~]$ ls -l /bin/usr 2> ls-error.txt
```

يتموضع رقم مقبض الملف "2" مباشرةً قبل معامل إعادة التوجيه لإعادة توجيه مجرى الخطأ القياسي إلى الملف ls-error.txt.

إعادة توجيه مجريي الخرج والخطأ إلى ملفٍ واحد

قد ترغب، في بعض الحالات، بإعادة توجيه مجريي الخرج والخطأ إلى ملفٍ واحد. لكنك ستحتاج إلى إعادة توجيه مجريي الخرج والخطأ في آن واحد. هنالك طريقتَين لفعل ذلك. الطريقة التقليدية التي تعمل مع الإصدارات القديمة من الصدفة هي:

```
[me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt 2>&1
```

نقوم في هذه الطريقة بعمليتَيّ إعادة توجيه. أولًا نُعيد توجيه مجرى الخرج القياسي إلى الملف ls-output.txt ومن ثم نُعيد توجيه مقبض الملف 2 (مجرى الخطأ القياسي) إلى مقبض الملف 1 (مجرى الخرج القياسي) باستخدام التعبير 2>&1.

لاحظ أن ترتيب عمليات إعادة التوجيه مهم. فإعادة توجيه مجرى الخطأ القياسي يجب أن تكون بعد إعادة توجيه مجرى الخرج القياسي. وإلا، فإن إعادة التوجيه لن تعمل كما يجب. المثال السابق:

```
>ls-output.txt 2>&1
```

يُعيد توجيه مجرى الخطأ القياسي إلى الملف ls-output.txt، لكن إذا تغير الترتيب إلى:

```
2>&1 >ls-output.txt
```

فسيبقى مجرى الخطأ القياسي مرتبطًا بالشاشة.

تحتوي الإصدارات الحديثة من صدفة bash على طريقة منظمة أكثر لإعادة التوجيه:

```
[me@linuxbox ~]$ ls -l /bin/usr &> ls-output.txt
```

استخدمنا في هذا المثال، معامل واحد فقط لإعادة توجيه مجريي الخرج والخطأ القياسيين إلى الملف ls-output.txt يُمكنك أيضًا أن تُلحِقَ ناتج مجريَيّ الخرج والخطأ القياسيَين إلى نفس الملف كالآتي:

```
[me@linuxbox ~]$ ls -l /bin/usr &>> ls-output.txt
```

التخلص من المخرجات

في بعض الأحيان يكون "السكوت من ذهب" كما يُقال، ولا نريد أيّة مخرجات من أمرٍ ما. أي نريد أن نهملها، وهذا ينطبق خصوصًا على رسائل الخطأ والحالة. يوفّر لنا النظام إمكانية ذلك بإعادة توجيه المخرجات إلى ملف خاص يُدعى "/dev/null". هذا الملف عبارة عن "جهاز" يقبل المدخلات ولا يفعل معها أي شيء. تستطيع تشبيه هذا الملف بالثقب الأسود. لكي نُسكِت رسائل الخطأ من أمرٍ ما، نقوم بعمليّة إعادة التوجيه الآتية:

```
[me@linuxbox ~]$ ls -l /bin/usr 2> /dev/null
```

/dev/null في ثقافة يونكس

إنّ /dev/null هو مفهوم قديم في يونكس؛ وبسبب شهرته، أُستخدم في العديد من المواضع في ثقافة يونكس. أصبحت تعرف ماذا يقصد أحدهم عندما يقول أنه يُرسل تعليقاتك إلى /dev/null. للمزيد من الأمثلة، راجع صفحة ويكيبيديا:

http://en.wikipedia.org/wiki//dev/null

إعادة توجيه مجرى الدخل القياسي

لم نستخدم إلى الآن أيّة أوامر تتعامل مع مجرى الدخل القياسي (في الواقع لقد استخدمناها، لكننا سنترك هذا الأمر ليكون مفاجأةً سنكشف عنها لاحقًا). لذا سنحتاج إلى تقديم أحدها.

لمّ الملفات باستخدام cat

يقرأ الأمر cat ملفًا واحدًا أو أكثر وينسخ محتواه إلى مجرى الخرج القياسي، كما في المثال الآتي:

```
cat [file...]
```

يمكنك تخيل أن الأمر cat شبيه بالأمر TYPE في نظام DOS.

تستطيع مشاهدة محتوى الملفات باستخدام cat بدون استخدام أحد البرامج كالبرنامج less. على سبيل

المثال، الأمر:

```
[me@linuxbox ~]$ cat ls-output.txt
```

سيعرض محتويات الملف ls-output.txt. يُستخدم الأمر cat عادةً لإظهار الملفات القصيرة نسبيًا. ولأن الأمر cat يقبل أكثر من ملف كوسيط؛ فيمكن استخدامه لِلمّ (أو ضم) الملفات مع بعضها. لنفترض أننا نزلنا من الإنترنت ملفًا كبيرًا جُزّءَ إلى عدّة أجزاء (تُقسّم ملفات الوسائط في شبكات USENET بهذه الطريقة)، ونريد لمّ تلك الملفات مع بعضها البعض. فإذا كانت الملفات مُسماة على الشكل:

```
movie.mpeg.001  movie.mpeg.002 ... movie.mpeg.099
```

فنستطيع لمّهم بالأمر:

```
cat movie.mpeg.0* > movie.mpeg
```

ولأن توسعة المحارف البديلة تكون مرتبة تصاعديًا؛ فسترتّب الوسائط ترتيبًا صحيحًا.

الأمور السابقة جيدة، لكن ما علاقتها بمجرى الدخل القياسي؟! ليس بعد. لكن لنجرب شيئًا آخر، ماذا لو استخدامنا الأمر cat دون وسائط:

```
[me@linuxbox ~]$ cat
```

لا يحدث أي شيء! فقط يتوقف كل شيء. لكن هذا ما يبدو عليه إلا أنه في الواقع يقوم بما عليه فعله.

سيقرأ الأمر cat المُدخلات من مجرى الدخل القياسي إن لم يُحدد معه أيّ وسيط. ولأن مجرد الدخل مرتبط افتراضيًا مع لوحة المفاتيح. فإنه في الواقع ينتظرنا لكي نكتب أي شيء! لذا جرب الآتي:

```
[me@linuxbox ~]$ cat
The quick brown fox jumped over the lazy dog.
```

الآن اضغط على الزرين Ctrl-d لتخبر الأمر cat أنه وصل إلى نهاية الملف (end of file أو اختصارًا EOF) في مجرى الدخل القياسي:

```
[me@linuxbox ~]$ cat
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
```

في حال لم يُحدَّد اسم لأحد الملفات كوسيط، فسيَنسخ الأمر cat مجرى الدخل القياسي إلى مجرى الخرج

القياسي؛ لذا، سنشاهد تكرار السطر الذي كتبناه. نستطيع استخدام هذا السلوك لإنشاء ملفات نصية قصيرة، لنفترض أننا نريد إنشاء ملف يُدعى "lazy_dog.txt" يحتوي على النص في المثال السابق:

```
[me@linuxbox ~]$ cat > lazy_dog.txt
The quick brown fox jumped over the lazy dog.
```

اطبع الأمر يليه النص الذي نريد أن نضعه في الملف. ولا تنس أن تضغط على Ctrl-d. لقد صنعنا أبسط محرر نصوص على الإطلاق! نستخدم الأمر cat مرّة أُخرى لننسخ الملف إلى مجرى الخرج القياسي كي نُشاهد محتوى الملف:

```
[me@linuxbox ~]$ cat lazy_dog.txt
The quick brown fox jumped over the lazy dog.
```

نعرف أن الأمر cat يقبل المدخلات من مجرى الدخل القياسي بالإضافة إلى الملفات. لذا، فلنجرب إعادة توجيه مجرى الدخل القياسي:

```
[me@linuxbox ~]$ cat < lazy_dog.txt
The quick brown fox jumped over the lazy dog.
```

لقد غيرّنا مصدر الدخل القياسي من لوحة المفاتيح إلى الملف lazy_dog.txt وذلك باستخدام المعامل "<". يُمكننا ملاحظة أن النتيجة هي نفسها عندما نُمرِّر اسم ملفٍ ما كوسيط؛ هذه العملية ليست مفيدة جدًا في هذا الصدد، لكنها تخدم غرض استخدام أحد الملفات كمصدر للدخل القياسي. هنالك أوامر أُخرى تُحقق فائدة أكبر من استخدام مجرى الدخل القياسي كما سنرى لاحقًا.

قبل أن نُكمل. تفقد صفحة man للأمر cat لأنها تحتوي على العديد من الخيارات المثيرة للاهتمام.

الأنابيب

تنتفع خاصية في الصدفة، تسمى "الأنابيب"، من خاصية القراءة من مجرى الدخل والإخراج إلى مجرى الخرج القياسيَين في أغلب الأوامر. عند استخدام المعامل الأنبوبي "|" (خط عمودي)، سيُمرر مجرى الخرج القياسي لأحد الأوامر إلى مجرى الدخل القياسي للأمر الآخر:

```
command1 | command2
```

سنحتاج إلى استخدام بعض الأوامر لكي نشرح هذه الفكرة بوضوح. هل تتذكر عندما قلنا أننا تعرفنا على أمر يقبل مجرى الدخل القياسي؟ إنه الأمر less. يُمكننا باستخدام الأمر less أن نعرض، صفحةً بصفحة،

مخرجات أي أمر يُرسلها إلى مجرى الدخل القياسي:

```
[me@linuxbox ~]$ ls -l /usr/bin | less
```

هذا الأمر رائع! نستطيع بكل أريحيـة، باستخدام هذه التقنيـة، أن نتفحص مخرجات أي أمر يُنتِج مخرجات تُرسَل إلى مجرى الخرج القياسي.

الفرق بين > و |

قد يبدو من الصعب فهم عملية إعادة التوجيه التي يقوم بها المعامل الأنبوبي "|" مقارنةً مع معامل إعادة التوجيه ">" من النظرة الأولى. ببساطة، يصل معامل إعادة التوجيه الأمر مع ملف، بينما يصل المعامل الأنبوبي بين مخرجات أحد الأوامر مع مدخلات أمرٍ آخر.

```
command1 > file1
command1 | command2
```

سيحاول الكثيرون تجربة الآتي عندما يتعلمون الأنابيب "كي يشاهدوا ما الذي سيحصل":

```
command1 > command2
```

الجواب: أحيانًا، شيءٌ سيءٌ للغاية!

هذا مثال حقيقي أُرسِل من أحد القرّاء الذي كان يدير خادم لينُكس. نُفِّذَ الآتي، كمستخدم جذر:

```
# cd /usr/bin
# ls > less
```

نقل أول أمر مجلد العمل الحالي إلى مجلد يُخزَّن فيه أغلب البرامج، وأخبر الأمر الثاني الصدفة بأن تُعيد الكتابة فوق الملف less مستبدلةً محتواه بمخرجات الأمر ls. ولأن المجلد usr/bin/ يحتوي على ملف باسم "less" (البرنامج less) ، فأدى الأمر الثاني إلى استبدال ملف البرنامج less بنص من مخرجات ls، مما أدى إلى تدمير برنامج less في نظامه!

الدرس الذي علينا تعلمه هو أن معامل إعادة التوجيه يُنشئ أو يُعيد الكتابة فوق الملفات بصمت. عليك أن تعامله بحذرٍ شديد.

المُرشِّحات

تُستخدم الأنابيب لإجراء عمليات معقدة على البيانات؛ حيث من الممكن استخدام أكثر من أمر معًا. تسمى عادةً الأوامر التي تعمل بهذا الشكل بالمُرشِحات أو الفلاتر. تأخذ المُرشِّحات المدخلات وتغيرها بشكلٍ ما ومن ثم تخرجها. أول أمر سنجرّبه هو أمر sort. لنفترض أننا نريد إنشاء قائمة بجميع الملفات التنفيذية الموجودة في المجلدين bin/ و usr/bin/، ومن ثم ترتيب النواتج وعرضها:

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | less
```

ولأننا قد حددنا مجلدَين (‎/bin‎ و ‎/usr/bin‎)، فستكون مخرجات الأمر ls قائمتين مرتبتين، قائمة لكل مجلد. وبتضمين الأمر sort في الأنبوب؛ فإننا سنُعدّل المخرجات لكي تعطي قائمة واحدة مرتبة.

التبليغ عن أو حذف الأسطر المكررة باستخدام الأمر uniq

عادةً ما يُستخدم الأمر uniq مع sort. يقبل الأمر uniq قائمةً مرتبةً إما من مجرى الدخل أو من ملف (راجع صفحة man للأمر uniq لمزيد من المعلومات) والذي يحذف افتراضيًّا أي سطر مكرر في القائمة. لذا، لكي نتأكد من أن قائمتنا لا تحتوي على أيّة سطور مكرّرة (تحمل بعض البرامج الموجودة في كلا المجلدين ‎/bin‎ و ‎/usr/bin‎ نفس الاسم) فسنستخدم الأمر uniq في الأنبوب:

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | less
```

استخدمنا، في المثال السابق، الأمر uniq لإزالة الأسطر المكررة. أما إذا أردنا معرفة الأسطر المكررة، فإننا نستخدم الخيار "d-" للأمر uniq:

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq -d | less
```

إظهار عدد الأسطر والكلمات والبايتات

يُستخدم الأمر wc (اختصار للكلمتَين word count) لحساب عدد الأسطر والكلمات والبايتات في الملفات. على سبيل المثال:

```
[me@linuxbox ~]$ wc ls-output.txt
  7902 64566 503634 ls-output.txt
```

طُبِعَ، في هذه الحالة، ثلاثة أرقام: عدد الأسطر وعدد الكلمات وعدد البايتات المحتواة في ملف ls-output.txt. وكما في الأوامر السابقة، سيعتمد الأمر wc على مجرى الدخل القياسي إن لم تُحدد أيّة وسائط. الخيار "l-" يجعل مخرجات الأمر wc مُقتصرةً فقط على عدد الأسطر. لمعرفة عدد البرامج الموجودة في القائمة لدينا، نستخدم الأمر الآتي:

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | wc -l
2728
```

طباعة الأسطر التي تُطابِق نمطًا معيناً باستخدام grep

الأداة grep هي أداة قوية في مطابقة أنماط نصيّة داخل الملفات. تُستخدم كالآتي:

```
grep pattern [file...]
```

عندما يواجه الأمر grep "نمطًا" في ملف، فسيعرض السطر الذي يحتويه. يمكن أن يكون النمط الذي يستطيع الأمر grep أن يُطابقه معقدًا للغاية. سنركز في الوقت الراهن على مطابقة النصوص البسيطة. وسنشرح الأنماط المتقدمة (تسمى التعابير النظامية [regular expressions]) في فصلٍ لاحق.

لنفترض أننا نريد البحث عن جميع الملفات (في قائمتنا المرتبة) التي تحتوي الكلمة "zip" في اسم الملف. مثل هذا البحث يعطينا فكرة عن البرمجيات الموجودة في نظام التشغيل لدينا التي تتعامل بشكلٍ أو بآخر مع ضغط الملفات. سيكون الأمر على الشكل الآتي:

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | grep zip
bunzip2
bzip2
gunzip
gzip
unzip
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
```

يوجد خياران مفيدان للأمر grep هما الخيار "i-" الذي يجعل grep يهمل اختلاف حالة الأحرف عند المطابقة (تكون عملية مطابقة الأنماط حساسة لحالة الأحرف افتراضيًّا) والخيار "v-" الذي يجعل grep يعرض الأسطر التي لا تطابق النمط.

طباعة بداية/نهاية الملفات باستخدام tail/head

لا نحتاج في بعض الأحيان إلى جميع مخرجات الأوامر. ربما نريد أول عدّة أسطر أو آخر عدّة أسطر. يطبع الأمر head أول عشرة أسطر ويطبع الأمر tail آخر عشرة أسطر. وكما لاحظتَ، يطبع الأمران head و tail عشرة أسطر افتراضيًّا؛ ويمكن تغيير ذلك بالخيار "n-":

```
[me@linuxbox ~]$ head -n 5 ls-output.txt
total 343496
-rwxr-xr-x 1 root root  31316 2007-12-05 08:58 [
-rwxr-xr-x 1 root root   8240 2007-12-09 13:39 411toppm
-rwxr-xr-x 1 root root 111276 2007-11-26 14:27 a2p
-rwxr-xr-x 1 root root  25368 2006-10-06 20:16 a52dec
[me@linuxbox ~]$ tail -n 5 ls-output.txt
-rwxr-xr-x 1 root root   5234 2007-06-27 10:56 znew
-rwxr-xr-x 1 root root    691 2005-09-10 04:21 zonetab2pot.py
-rw-r--r-- 1 root root    930 2007-11-01 12:23 zonetab2pot.pyc
-rw-r--r-- 1 root root    930 2007-11-01 12:23 zonetab2pot.pyo
lrwxrwxrwx 1 root root      6 2008-01-31 05:22 zsoelim -> soelim
```

وبالطبع يمكن استخدامهما أيضًا في الأنابيب:

```
[me@linuxbox ~]$ ls /usr/bin | tail -n 5
znew
zonetab2pot.py
zonetab2pot.pyc
zonetab2pot.pyo
zsoelim
```

لدى الأمر `tail` خيار يسمح بمراقبة الملفات في الوقت الحقيقي. قد يكون هذا الأمر مُفيدًا عند مراقبة أحد ملفات السجلات (log) في أثناء الكتابة إليه. سنلقي نظرة، في المثال الآتي، على ملف `messages` في المجلد `/var/log` (أو الملف `/var/log/syslog` إذا لم يكن الملف `messages` موجودًا). بعض التوزيعات تتطلب امتيازات الجذر لكي تتمكن من مشاهدة هذا الملف لأنه قد يحتوي على معلومات أمنية:

```
[me@linuxbox ~]$ tail -f /var/log/messages
Feb 8 13:40:05 twin4 dhclient: DHCPACK from 192.168.1.1
Feb 8 13:40:05 twin4 dhclient: bound to 192.168.1.4 -- renewal in  1652
seconds.
Feb 8 13:55:32 twin4 mountd[3953]: /var/NFSv4/musicbox exported to
both 192.168.1.0/24 and twin7.localdomain in
192.168.1.0/24,twin7.localdomain
Feb 8 14:07:37 twin4 dhclient: DHCPREQUEST on eth0 to 192.168.1.1
port 67
```

```
Feb 8 14:07:37 twin4 dhclient: DHCPACK from 192.168.1.1
Feb 8 14:07:37 twin4 dhclient: bound to 192.168.1.4 -- renewal in
1771 seconds.
Feb 8 14:09:56 twin4 smartd[3468]: Device: /dev/hda, SMART
Prefailure Attribute: 8 Seek_Time_Performance changed from 237 to 236
Feb 8 14:10:37 twin4 mountd[3953]: /var/NFSv4/musicbox exported to
both 192.168.1.0/24 and twin7.localdomain in
192.168.1.0/24,twin7.localdomain
Feb 8 14:25:07 twin4 sshd(pam_unix)[29234]: session opened for user
me by (uid=0)
Feb 8 14:25:36 twin4 su(pam_unix)[29279]: session opened for user
root by me(uid=500)
```

سيستمر الأمر `tail` بمراقبة الملف عند استخدام الخيار `-f`. وستُعرض الأسطر الجديدة فورًا عند إضافتها. وسيستمر ذلك إلى أن يضغط المستخدم على `Ctrl-c`.

القراءة من مجرى الدخل والكتابة إلى مجرى الخرج وإلى الملفات

لنبق الآن داخل "أنابيبنا". يوفر نظام لينُكس أمرًا باسم tee. يقرأ الأمر tee من مجرى الدخل القياسي وينسخه إلى مجرى الخرج القياسي (ليسمح للبيانات بمواصلة العبور في الأنبوب) وإلى ملف واحد أو أكثر. يُفيد هذا الأمر في التقاط محتوى الأنبوب في منتصف مرحلة المعالجة. سنُكرر الآن أحد الأمثلة السابقة. لكن هذه المرة مع استخدام tee لأخذ نسخة من القائمة التي تحتوي على ملفات المجلد ووضعها في ملف `ls.txt` قبل أن يُرشِّحها grep:

```
[me@linuxbox ~]$ ls /usr/bin | tee ls.txt | grep zip
bunzip2
bzip2
gunzip
gzip
unzip
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
```

الخلاصة

كالعادة، راجع الدليل لكل من الأوامر التي شُرِحَت في هذا الفصل. لقد شاهدنا الاستخدامات البسيطة لهذه الأوامر فقط. لدى كل تلك الأوامر العديد من الخيارات المثيرة للاهتمام. سندرك مدى أهمية إعادة التوجيه لحل المشكلات المعقدة كلما ازدادت خبرتنا في لينُكس. توجد العديد من الأوامر التي تستخدم مجريَيِ الدخل والخرج القياسيَين. تستخدم جميع الأوامر تقريبًا مجرى الخطأ القياسي لعرض رسائل الخطأ الخاصة بها.

لينُكس نظام متعلق بالتخيلات

عندما أُسأل عن الفروقات ما بين لينُكس وويندوز. فإني غالبًا ما أستخدم نظرية الدمى.

يُشبه نظام ويندوز دمية على شكل صبي. فأنت تذهب إلى المتجر وتشتري أكثر دمية براقة وجديدة من الصندوق. تأخذها إلى المنزل وتُشغِّلها وتلعب معها؛ حيث تتمتع برسوميات جميلة وأصوات رائعة. لكن بعد فترة قصيرة ستبدأ بالملل منها وتعود إلى المتجر وتشتري دميةً أُخرى. وتعود العودة مرارًا وتكرارًا حتى تقرر أخيرًا أن تعود إلى المتجر وتقول للبائع "أريد لعبة تقوم بهذا الشيء" وتجده يُخبرك أنه لا يوجد هكذا دمية لأنه لا يوجد طلب عليها. ثم تقول "ولكنني أريد أن أغير هذا الشيء!" ويقول لك البائع أنه ليس بإمكانك تغييره. ثم تكتشف أن دميتك مرهونة بالأشياء التي قرر الآخرون أنك ستحتاج إليها ولا أكثر من ذلك.

أما لينُكس، فهو أكبر مجموعة قطع ألعاب في العالم، ستفتح العلبة وستجد مجموعة كبيرة من أجزاء الدمى. الكثير من الدعائم، والبراغي، والحزقات، والمسننات، والبكرات، والمحركات وبعض الاقتراحات عن الدمية التي ستبنيها. لذا، عندما تبدأ اللعب فيه، فستبني أحد تلك الاقتراحات، ثم تنتقل للآخر. ثم بعد فترة، تدرك أنك تستطيع إنشاء أفكارك الخاصة. لن تعود إلى المتجر أبدًا، لأن لديك كل ما تريد.

الخيار عائدٌ لك. أيّة دمية تستحق أن تلعب بها؟

الفصل السابع:

رؤية العالم كما تراه الصدفة

سنلقي نظرة، في هذا الفصل، على بعض "السحر" الذي يحدث في سطر الأوامر عند الضغط على زر Enter. وعلى الرغم من أننا سنناقش العديد من الميزات المسلية والمعقدة المتعلقة بالصدفة، إلا أننا سنفعل ذلك بأمر جديد وحيد:

- echo – عرض سطر نصيّ.

التوسعة

في كل مرة نطبع فيها أمرًا ونضغط على رز Enter، فإن bash تقوم بعدّة عمليات عليه قبل أن تُنَفِّذه. شاهدنا حالتَين من الحالات كيف يحتوي حرف بسيط، على سبيل المثال الحرف "*"، على معانٍ كثيرة للصدفة. العملية التي تقوم بذلك تسمى "التوسعة" (Expansion). مع التوسعة، فإنك تكتب شيئًا ما ويتوسع إلى شيء آخر قبل أن تنفذه الصدفة. لنلقِ نظرة على الأمر echo لشرح المعنى الذي نعنيه. echo هو أمر مُضمّن في الصدفة يقوم بمهمة بسيطة للغاية ألا وهي طباعة الوسائط النصية المُمررة إليه إلى مجرى الخرج القياسي:

```
[me@linuxbox ~]$ echo this is a test
this is a test
```

استخدامه سهل للغاية، حيث يُظهر أيّ وسيط يُمرر إليه. لنجرب مثالًا آخر:

```
[me@linuxbox ~]$ echo *
Desktop Documents ls-output.txt Music Pictures Public Templates
Videos
```

ماذا حدث؟ لماذا لم يطبع الأمر echo الرمز "*"؟ كما تتذكر عند تعاملنا مع المحارف البديلة، فإن الرمز "*" يُطابق أي محرف في اسم الملف. لكننا لم نذكر في نقاشنا الأصلي كيف تقوم الصدفة بذلك. الجواب المُبسّط هو أن الصدفة توسع الرمز "*" إلى شيء آخر (في هذا المثال، أسماء الملفات الموجودة في مجلد العمل الحالي) قبل تنفيذ الأمر echo. فعند الضغط على زر Enter، توسّع الصدفة تلقائيًا أيّة محارف قابلة للتوسيع قبل تنفيذ الأمر. لذا، لن يرى الأمر echo الرمز "*" أبدًا، فقط النتيجة الموسعة. نجد أن الأمر echo سلك سلوكه الطبيعي بعد معرفتنا لهذه المعلومات.

توسعة أسماء الملفات

الآلية التي تعمل بها المحارف البديلة تسمى "توسعة أسماء الملفات". إذا جربنا بعض التقنيات التي تعلمناها في الفصول الأولى، فسندرك أنها فعلًا توسعة! فلنفرض أن مجلد المنزل لدينا يحتوي الآتي:

```
[me@linuxbox ~]$ ls
Desktop     ls-output.txt   Pictures   Templates
Documents   Music           Public     Videos
```

سننفذ التوسعات الآتية:

```
[me@linuxbox ~]$ echo D*
Desktop Documents
```

والأمر:

```
[me@linuxbox ~]$ echo *s
Documents Pictures Templates Videos
```

أو حتى:

```
[me@linuxbox ~]$ echo [[:upper:]]*
Desktop Documents Music Pictures Public Templates Videos
```

وحتى لو خرجنا عن مجلد المنزل:

```
[me@linuxbox ~]$ echo /usr/*/share
/usr/kerberos/share /usr/local/share
```

توسعة أسماء الملفات المخفية

كما تعلم، إن أسماء الملفات التي تبدأ بنقطة هي ملفات مخفية. آلية توسعة أسماء الملفات تحترم ذلك. فتوسعة كالآتية:

```
echo *
```

لا تُظهر الملفات المخفية.

قد يبدو للوهلة الأولى أننا نستطيع تضمين الملفات المخفية في التوسعة ببداية النمط بنقطة، كالآتي:

```
echo .*
```

ستجد -وللوهلة الأولى- أنه عمل بنجاح. لكن إذا دققت بالنتائج فستجد أن "." و ".." يظهران في النتائج أيضًا. ولأن هذين الاسمين يُشيران إلى المجلد الحالي والمجلد الأب، فإن استخدام هذا النمط سيولد في أغلب الحالات نتائج مغلوطة. نستطيع مشاهدة ذلك إذا جربنا الأمر الآتي:

```
ls -d .* | less
```

يجب علينا تحديد نمط أكثر دقةً وتحديدًا للقيام بعملية توسعة لأسماء الملفات بشكل صحيح. هذا الأمر سيعمل على ما يرام:

```
ls -d .[!.]?*
```

سيُوسّع هذا النمط إلى اسم كل ملف يبدأ بنقطة ولا تتبعه نقطة أُخرى ويحتوي على محرف آخر إضافي ويمكن أن يتبعه أي عدد من المحارف. سيعمل الأمر السابق بنجاح مع أغلب الملفات المخفية (لكنه لا يُضمِّن أسماء الملفات التي تبدأ بعدّة نقط). سيوفر الأمر ls مع الخيار A- قائمةً صحيحةً تتضمن الملفات المخفية:

```
ls -A
```

توسعة رمز المدّة

كما تتذكر من بداياتنا مع أمر cd، فإن للرمز "~" معنى خاص. سيُوسّع الرمز "~" عندما يُستخدم قبل اسم أيّ مستخدم إلى مجلد المنزل الخاص بذاك المستخدم. أما إذا لم يُحدد اسم المستخدم، فسيُوسّع إلى مسار مجلد المنزل للمستخدم الحالي:

```
[me@linuxbox ~]$ echo ~
/home/me
```

إذا كان لدينا مستخدم باسم "foo"، فإن:

```
[me@linuxbox ~]$ echo ~foo
/home/foo
```

توسعة العمليات الحسابية

تسمح الصدفة بالقيام بالعمليات الحسابية بواسطة التوسعة. وهذا ما يُمكّننا من استخدام الصدفة كآلة حاسبة:

```
[me@linuxbox ~]$ echo $((2 + 2))
4
```

الشكل العام لتوسعة التعابير الحسابية هو:

```
$((expression))
```

حيث "expression" هو عملية حسابية تحتوي على قيم عددية ومعاملات رياضية.

لا تدعم التعابير الحسابية سوى الأعداد الصحيحة (الأعداد الموجبة والسالبة ولا تحتوي على فاصلة عشرية)، لكنها تستطيع القيام بعدد كبير من العمليات الرياضية المختلفة. يحتوي هذا الجدول على عددٍ من المعاملات المدعومة:

الجدول 7-1: المعاملات الحسابية

المعامل	الشرح
+	الجمع.
-	الطرح.
*	الضرب.
/	القسمة (لكن تذكر أن العمليات الحسابية تجري فقط على الأعداد الصحيحة).
%	باقي القسمة.
**	الرفع إلى الأس.

الفراغات غير مهمة في التعابير الحسابية، ويمكن استخدام الأقواس مع التعابير. على سبيل المثال، لضرب خمسة مربع في ثلاثة، نستخدم الأمر الآتي:

```
[me@linuxbox ~]$ echo $(($((5**2)) * 3))
75
```

ويمكن للأقواس الأحادية أن تجمع عدة تعابير فرعية. نستطيع باستخدام هذه الطريقة إعادة كتابة المثال السابق بعملية توسعة واحدة والحصول على نفس النتيجة:

```
[me@linuxbox ~]$ echo $(((5**2) * 3))
75
```

يستخدم المثال الآتي مُعاملَيّ القسمة وباقي القسمة. لاحظ أثر القسمة في الأعداد الصحيحة:

```
[me@linuxbox ~]$ echo Five divided by two equals $((5/2))
Five divided by two equals 2
[me@linuxbox ~]$ echo with $((5%2)) left over.
with 1 left over.
```

ستُشرَح توسعة العمليات الحسابية بالتفصيل في الفصل 34.

توسعة الأقواس

ربما تكون توسعة الأقواس من أكثر عمليات التوسعة غرابةً. نستطيع بواسطتها أن نُنشئ سلاسل نصية متعددة من نمط واحد يحتوي على الأقواس، هذا مثال عنها:

```
[me@linuxbox ~]$ echo Front-{A,B,C}-Back
Front-A-Back Front-B-Back Front-C-Back
```

الأنماط التي تحتوي على توسعة أقواس قد تحتوي على عبارة تمهيدية تسمى المقدمة (preamble)، وعبارة ختامية تسمى الحاشية (postscript). تحتوي الأقواس إما على قائمة تتكون من سلاسل نصية مفصولة بفواصل ","، أو على مجال من الأرقام أو الأحرف الأبجدية. ولا يمكن أن يحتوي النمط على فراغات. هذا مثال عن استخدام مجال للأرقام:

```
[me@linuxbox ~]$ echo Number_{1..5}
Number_1 Number_2 Number_3 Number_4 Number_5
```

ويمكن استخدام الصفر قبل الأعداد كالآتي:

```
[me@linuxbox ~]$ echo {01..15}
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
[me@linuxbox ~]$ echo {001..15}
001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
```

وهذا مثال يعرض الأحرف الأبجدية بترتيب معكوس:

```
[me@linuxbox ~]$ echo {Z..A}
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

ويمكن للأقواس القابلة للتوسعة أن تحتوي على أقواس أُخرى وهكذا:

```
[me@linuxbox ~]$ echo a{A{1,2},B{3,4}}b
aA1b aA2b aB3b aB4b
```

حسنًا، بماذا تفيد هذه الأشياء؟ أشهر التطبيقات هو إنشاء قائمة بأسماء الملفات أو المجلدات التي ستُنشأ. على سبيل المثال، إذا كنا مصورين فوتوغرافيين وكانت لدينا مجموعة ضخمة من الصور ونريد تنظيمها حسب السنة والشهر؛ فإن أول ما سنقوم به هو إنشاء سلسلة من المجلدات تسمى وفق النمط "Year-Month". ستُرتب المجلدات في هذه الطريقة ترتيبًا زمنيًا. بإمكاننا بالطبع أن نكتب قائمة المجلدات كاملةً يدويًا لكن ذلك سيحتاج إلى الكثير من العمل إضافةً إلى إمكانية وقوع أخطاء. فبدلًا من ذلك، يُمكننا تنفيذ الأوامر الآتية:

```
[me@linuxbox ~]$ mkdir Pics
[me@linuxbox ~]$ cd Pics
[me@linuxbox Photos]$ mkdir {2007..2009}-{01..12}
[me@linuxbox Photos]$ ls
2007-01  2007-07  2008-01  2008-07  2009-01  2009-07
2007-02  2007-08  2008-02  2008-08  2009-02  2009-08
2007-03  2007-09  2008-03  2008-09  2009-03  2009-09
2007-04  2007-10  2008-04  2008-10  2009-04  2009-10
2007-05  2007-11  2008-05  2008-11  2009-05  2009-11
2007-06  2007-12  2008-06  2008-12  2009-06  2009-12
```

طريقة رائعة!

توسعة المعاملات

سنشرح توسعة المعاملات باختصار في هذا الفصل، إلا أننا سنشرحها شرحًا مكثفًا لاحقًا. توسعة المعاملات هي ميزة مفيدة في السكربتات أكثر منها في سطر الأوامر مباشرةً. تتعامل معظم إمكانياتها مع مقدرة النظام على تخزين قطع صغيرة من البيانات وقابلية إعطاء اسم لها. العديد من القطع -عادةً ما يُشار إليها بالمتغيرات- تسمح لك بفحص محتواها. على سبيل المثال، المعامل المُسمى "USER" يحتوي على اسم المستخدم الخاص بك. تُستخدم الطريقة الآتية لتوسيع المعاملات والحصول على محتويات المتغير USER:

```
[me@linuxbox ~]$ echo $USER
me
```

جرّب الأمر الآتي كي تشاهد قائمة بجميع المتغيرات المتوفرة:

```
[me@linuxbox ~]$ printenv | less
```

ربما لاحظت في باقي أنواع التوسعات أنك إذا أخطأت في النمط فإن التوسعة لا تتم ويُظهر الأمر echo النمط الذي يحتوي على أخطاء. أما عند الخطأ في توسعة المعاملات، فتتم التوسعة لكن الناتج هو سلسلة نصية فارغة:

```
[me@linuxbox ~]$ echo $SUER
[me@linuxbox ~]$
```

تعويض الأوامر

يسمح تعويض الأوامر باستخدام ناتج أمرٍ ما كعملية توسعة:

```
[me@linuxbox ~]$ echo $(ls)
Desktop Documents ls-output.txt Music Pictures Public Templates
Videos
```

وواحد من الأوامر المفضلة عندي هو:

```
[me@linuxbox ~]$ ls -l $(which cp)
-rwxr-xr-x 1 root root 71516 2007-12-05 08:58 /bin/cp
```

لقد مررنا ناتج الأمر "which cp" كوسيط إلى الأمر ls. وهذا يؤدي إلى الحصول على معلومات البرنامج cp دون أن نعرف المسار الكامل له. لسنا محدودين بالأوامر البسيطة. فبإمكاننا استخدام ناتج مخرجات الأنبوب بأكمله (يُعرَض في المثال الآتي جزء من النواتج فقط):

```
[me@linuxbox ~]$ file $(ls /usr/bin/* | grep zip)
/usr/bin/bunzip2:  symbolic link to `bzip2'
/usr/bin/bzip2:    ELF 32-bit LSB executable, Intel 80386,
version 1 (SYSV), dynamically linked (uses shared libs), for
GNU/Linux 2.6.9, stripped
/usr/bin/bzip2recover: ELF 32-bit LSB executable, Intel 80386,
version 1 (SYSV), dynamically linked (uses shared libs), for
GNU/Linux 2.6.9, stripped
/usr/bin/funzip:   ELF 32-bit LSB executable, Intel 80386,
version 1 (SYSV), dynamically linked (uses shared libs), for
GNU/Linux 2.6.9, stripped
/usr/bin/gpg-zip:  Bourne shell script text executable
```

```
/usr/bin/gunzip:   symbolic link to `../../bin/gunzip'
/usr/bin/gzip:     symbolic link to `../../bin/gzip'
/usr/bin/mzip:     symbolic link to `mtools'
```

أُستخدم، في المثال السابق، ناتج الأنبوب كوسيط للأمر `file`.

توجد هنالك صياغة أُخرى لتعويض الأوامر في الصدفات القديمة، وما تزال تلك الصياغة مدعومةً في bash. تَستخدم تلك الصياغة علامات الاقتباس الخلفية (موجودة فوق زر tab) بدلًا من إشارة الدولار والأقواس:

```
[me@linuxbox ~]$ ls -l `which cp`
-rwxr-xr-x 1 root root 71516 2007-12-05 08:58 /bin/cp
```

الاقتباس

لقد شاهدنا العديد من الطرق التي تقوم الصدفة فيها بالتوسعة. حان الوقت الآن لتعلم كيفية التحكم فيها. على سبيل المثال:

```
[me@linuxbox ~]$ echo this is a      test
this is a test
```

أو:

```
[me@linuxbox ~]$ echo The total is $100.00
The total is 00.00
```

في المثال الأول، يُزيل "تقسيم الكلمات" الذي تقوم به الصدفة الفراغات الزائدة من وسائط الأمر echo. في المثال الثاني، قامت توسعة المتغيرات بوضع سلسلة نصية فارغة مكان المتغير "$1" لأن المتغير غير معرّف. تُوفّر الصدفة آلية تدعى "الاقتباس" كي نختار التوسعات التي نريد إيقافها.

الاقتباس المزدوج

أول نوع من أنواع الاقتباس التي سنناقشها هو الاقتباس المزدوج. إذا وضعت نصًا داخل علامتَيّ اقتباس مزدوجتَين، فستفقد جميع المحارف الخاصة في الصدفة معناها وتُعامَل كمحارف عادية. الاستثناءات هي "$"و "\" (الشرطة المائلة الخلفية) و "``" (علامة الاقتباس الخلفية). هذا يعني أن تقسيم الكلمات وتوسعة أسماء الملفات وتوسعة رمز المّدة وتوسعة الأقواس سيتم تجاهلها تمامًا. لكن توسعة المتغيرات والعمليات الرياضية وتعويض الأوامر سيتم القيام بها. نستطيع استخدام أسماء الملفات التي تحتوي على فراغات

بوضعها بين علامتَيّ اقتباس مزدوجتَين. لنفترض أننا ضحية غير محظوظة لملف اسمه two words.txt. إذا حاولنا تجربته في أمرٍ ما؛ فسيؤدي تقسيم الكلمات إلى معاملته على أنه وسيطَين منفصلَين بدلًا من وسيط واحد:

```
[me@linuxbox ~]$ ls -l two words.txt
ls: cannot access two: No such file or directory
ls: cannot access words.txt: No such file or directory
```

نستطيع إيقاف تقسيم الكلمات والحصول على النتيجة المطلوبة باستخدام علامتَيّ الاقتباس المزدوجتَين. وحتى أننا نستطيع تصحيح اسم الملف:

```
[me@linuxbox ~]$ ls -l "two words.txt"
-rwxr-xr-x 1 me me 124932 Jan 17 2013 two words.txt
[me@linuxbox ~]$ mv "two words.txt" two_words.txt
```

تستطيع الآن الاستغناء عن كتابة علامات الاقتباس!

تذكر أن توسيع المتغيرات والعمليات الحسابية وتعويض الأوامر ما زالت فعّالة عند استخدام علامتَي الاقتباس المزدوجتَين:

```
[me@linuxbox ~]$ echo "$USER $((2+2)) $(cal)"
me 4    February 2008
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29
```

علينا أن نلقي نظرة على تأثير علامتي الاقتباس المزدوجتين على تعويض الأوامر. لكن أولًا، لنلق نظرة معمّقة على طريقة تقسيم الكلمات. في أحد الأمثلة السابقة، رأينا كيف يحذف تقسيم الكلمات الفراغات الزائدة في النص:

```
[me@linuxbox ~]$ echo this is a    test
this is a test
```

افتراضيًّا، يبحث تقسيم الكلمات عن وجود الفراغات أو مسافات الجدولة أو الأسطر الجديدة ويعاملهم على

أنهم "فواصل" بين الكلمات. هذا يعني أنه في النص غير المحاط بعلامات اقتباس، لا تُعتَبر الفراغات ومسافات الجدولة والأسطر الجديدة على أنها جزء من النص. يحتوي مثالنا السابق على اسم الأمر يتبعه أربعة وسائط مختلفة. أما إذا أضفنا علاماتَيّ الاقتباس المزدوجتَين:

```
[me@linuxbox ~]$ echo "this is a    test"
this is a    test
```

فقد أُوقِف تقسيم الكلمات ولم تُعامَل الفرغات على أنها "فواصل"، بل أصبحت جزءًا من الوسيط. فأصبح الأمر يحتوي على وسيط واحد وذلك عند استخدام علامتي الاقتباس.

في الواقع، إن اعتبار الأسطر الجديدة فاصلًا في آلية تقسيم الكلمات يؤثر على تعويض الأوامر. جرب المثال الآتي:

```
[me@linuxbox ~]$ echo $(cal)
February 2008 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29
[me@linuxbox ~]$ echo "$(cal)"
   February 2008
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29
```

أُعتُبِرَ، في الأمر الأول، ناتج تعويض الأمر cal ثمان وثلاثون وسيطًا. لكن أُعتبر الناتج وسيطًا وحيدًا يحتوي على فراغات وأسطر جديدة في المثال الثاني.

الاقتباس الفردي

إذا لم نُرِد إجراء أيّة عملية توسعة في الأمر، فيجب علينا استخدام علامات الاقتباس المفردة. هذه مقارنة بين ثلاثة أوامر، الأول دون أي علامات اقتباس والثاني بعلامتي اقتباس مزدوجتين والثالث بعلامتي اقتباس مفردتين:

```
[me@linuxbox ~]$ echo text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
text /home/me/ls-output.txt a b foo 4 me
[me@linuxbox ~]$ echo "text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER"
```

```
text ~/*.txt {a,b} foo 4 me
[me@linuxbox ~]$ echo 'text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER'
text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
```

كما لاحظنا، ستتجاهل الصدفة العديد من التوسعات كلما ازدادت "درجة" الاقتباس.

تهريب المحارف

قد تريد في بعض الأحيان أن تقتبس محرفًا واحدًا فقط. يُمكننا للقيام بذلك بإسباق المحرف المراد "تهريبه" برمز الشرطة المائلة الخلفية، الذي يُسمى في هذا السياق بمحرف الهروب. عادة ما يُستخدم داخل علامات الاقتباس المزدوجة كي يَمنع تنفيذ توسعة ما:

```
[me@linuxbox ~]$ echo "The balance for user $USER is: \$5.00"
The balance for user me is: $5.00
```

استخدمنا الشرطة المائلة الخلفية لإزالة معنى المحارف الخاصة في أسماء الملفات. على سبيل المثال، من الممكن استخدام بعض المحارف التي تملك معنى خاص للصدفة. هذه المحارف تتضمن: "$" و "!" و "&" و " " وغيرها. نستخدم التقنية الآتية لتضمين حرف خاص في اسم الملف:

```
[me@linuxbox ~]$ mv bad\&filename good_filename
```

للسماح للشرطة المائلة الخلفية بالظهور، فإننا نقوم "بتهريبها" بطباعة "\\". لاحظ أن الشرطة المائلة الخلفية تفقد معناها عند استخدام علامتي الاقتباس المفردتين وتُعامَل كأي محرف عادي.

"سلاسل الهروب" باستخدام الشرطة المائلة الخلفية

بالإضافة إلى دورها كمحرف للهروب، تلعب الشرطة المائلة الخلفية دورًا في تمثيل بعض المحارف الخاصة التي تسمى بأكواد التحكم. يُستخدَم أول 32 محرفًا من أكواد ASCII لنقل الأوامر لأجهزة شبيهة بالتلغراف. بعض تلك الأكواد مألوف (مسافة الجدولة [tab] والفراغ الخلفي [backspace] ومحرف السطر الجديد [linefeed]، ومحرف العودة إلى بداية السطر [carriage return])، بينما بعضها الآخر غير مشهور (اللاشيء [null]، نهاية الإرسال).

سلسلة الهروب	الشرح
\a	الجرس ("التنبيه" - يؤدي إلى أن يزمر الحاسوب).
\b	الفراغ الخلفي (backspace).

\n	محرف السطر الجديد.
\r	محرف العودة إلى بداية السطر.
\t	مسافة جدولة (tab).

يحتوي الجدول السابق على بعض سلاسل الهروب الشهيرة. السبب في هذا التمثيل هو أن الشرطة المائلة الخلفية تنحدر أصولها من لغة برمجة C وتُعتمد من قِبل الكثيرين، بما فيهم الصدفة.

ستؤدي إضافة الخيار "e-" إلى أمر echo إلى تفعيل تفسير سلاسل الهروب. ويمكنك أيضًا وضعها داخل التعبير `$' '`. باستخدام الأمر sleep، الذي يُمثل برنامجًا صغيرًا ينتظر لعدد من الثواني ومن ثم ينتهي تنفيذه؛ سنُنشئ مؤقت عدّ تنازلي باستخدام الأمر الآتي:

```
sleep 10; echo -e "Time's up\a"
```

نستطيع كتابة ذاك الأمر كالآتي:

```
sleep 10; echo "Time's up" $'\a'
```

الخلاصة

لقد تحسن مستوانا كثيرًا في استخدام الصدفة، سنجد أن استخدام التوسعات والاقتباسات شائع جدًا. لذا، من المفيد أن تفهم آلية عملها فهمًا جيدًا. في الحقيقة، يمكن اعتبار التوسعات والاقتباسات من أهم المواضيع التي يجب تعلمها. ستكون التوسعات مصدرًا للغموض وللإرباك إذا لم تُفهَم جيدًا، عدا عن إهدار العديد من الإمكانيات التي تتحلى بها الصدفة.

الفصل الثامن:

استخدامات متقدمة للوحة المفاتيح

غالبًا ما أصف مازحًا نظام يونكس بأنه "نظام التشغيل للأشخاص الذين يحبون الطباعة". وعلى الرغم من أن سطر الأوامر يبرهن ذاك الكلام، إلا أن مستخدمي سطر الأوامر لا يحبون الطباعة لهذه الدرجة. لماذا إذًا تكون أسماء العديد من الأوامر قصيرة للغاية كالأوامر `cp` و `ls` و `mv` و `rm`؟ في الحقيقة، إن أحد أكثر أهداف سطر الأوامر المثيرة للاهتمام هو الكسل؛ القيام بمعظم الأعمال بمجرد ضغطات بسيطة على لوحة المفاتيح. هدف آخر هو أن لا ترفع أصابعك عن لوحة المفاتيح؛ إلى درجة الاستغناء عن استخدام الفأرة! سنناقش في هذا الفصل ميزات الصدفة bash التي تجعل استخدام لوحة المفاتيح أكثر سرعةً وفعاليةً.

ستُستخدم الأوامر الآتية في هذا الفصل:

- `clear` – مسح محتويات الشاشة.
- `history` – عرض محتويات قائمة تأريخ الأوامر.

التعديلات في سطر الأوامر

تستخدم bash مكتبة (مجموعة مشتركة من الأكواد التي تقوم بأعمال معينة تستخدمها مختلف البرامج) تسمى `Readline` لكي تحقق إمكانية التعديل في سطر الأوامر. لقد جربنا ذلك مسبقًا عند استخدام أزرار الأسهم لتحريك المؤشر؛ لكن يوجد هناك المزيد من الميزات التي لم نجربها بعد. يمكنك اعتبارها أدوات إضافية تساعدك في عملك. صحيح أنه ليس من الضروري تعلمها جميعًا؛ إلا أن غالبيتها تكون ذات استخدامات مفيدة. اختر منها ما تشاء!

ملاحظة: يمكن أن تُفسَّر بعض الاختصارات التي ستُذكر (وخصوصًا تلك التي تستخدم الزر Alt) من قِبل الواجهة الرسومية لتقوم بمهام أُخرى. يجب أن تعمل جميع الاختصارات المذكورة جيدًا مع الطرفية الوهمية.

تحريك المؤشر

يحتوي الجدول الآتي على قائمة بالأزرار أو الاختصارات التي تُستخدم لتحريك المؤشر:

الجدول 8-1: أوامر نقل المؤشر

الزر	الشرح
Ctrl-a	نقل المؤشر إلى بداية السطر.
Ctrl-e	نقل المؤشر إلى نهاية السطر.
Ctrl-f	نقل المؤشر إلى الأمام بمقدار حرف واحد؛ نفس تأثير استخدام السهم الأيمن.
Ctrl-b	نقل المؤشر إلى الخلف بمقدار حرف واحد؛ نفس تأثير استخدام السهم الأيسر.
Alt-f	نقل المؤشر كلمة واحدة إلى الأمام.
Alt-b	نقل المؤشر كلمة واحدة إلى الخلف.
Ctrl-l	مسح محتويات الشاشة وتحريك المؤشر إلى الزاوية العليا اليسرى من الشاشة. يقوم الأمر clear بنفس المهمة.

تعديل النص

يعرض الجدول الآتي الاختصارات المُستخدَمة لتعديل المحارف في سطر الأوامر:

الجدول 8-2: أوامر تعديل النص

الزر	الشرح
Ctrl-d	حذف الحرف الموجود عند المؤشر.
Ctrl-t	استبدال الحرف الموجود عند المؤشر بالحرف الذي قبله.
Alt-t	استبدال الكلمة الموجودة عند المؤشر بالكلمة التي قبلها.
Alt-l	تحويل حالة جميع الحروف من مكان وجود المؤشر إلى نهاية الكلمة إلى حالة الأحرف الصغيرة (lowercase).
Alt-u	تحويل حالة جميع الحروف من مكان وجود المؤشر إلى نهاية الكلمة إلى حالة الأحرف الكبيرة (uppercase).

قص ولصق النصوص

يستخدم التوثيق الخاص بمكتبة Readline المصطلحَين killing و yanking للإشارة إلى القص واللصق كما هو متداول حاليًا. توضع العناصر التي تُقص في حافظة تسمى kill-ring.

الجدول 3-8: أوامر القص واللصق

الزر	الشرح
`Ctrl-k`	قص النص من موضع المؤشر إلى نهاية السطر.
`Ctrl-u`	قص النص من موضع المؤشر إلى بداية السطر.
`Alt-d`	قص النص من موضع المؤشر إلى نهاية الكلمة.
`Alt-Backspace`	قص النص من مكان المؤشر إلى بداية الكلمة الحالية؛ إذا كان المؤشر في بداية الكلمة فستُقص الكلمة السابقة.
`Ctrl-y`	لصق النص من حافظة kill-ring وإدراجه في مكان وجود المؤشر.

ما هو زر Meta؟

إذا أقدمتَ على قراءة توثيق Readline، الذي تستطيع العثور عليه في قسم READLINE في صفحة bash في دليل man، فإنك ستواجه المصطلح "Meta key"، الذي يُشير في الحواسيب الحديثة إلى الزر Alt؛ لكنه لم يكن كذلك دائمًا.

في العصور المظلمة (قبل الحواسيب الشخصية لكن بعد ظهور يونكس)، لم يكن يملك كل فرد حاسبًا شخصيًّا. الجهاز الذي قد يملكونه كان يُسمى "طرفية" (terminal). الطرفية هي جهاز للتواصل مع الحاسوب يحتوي على شاشة ولوحة مفاتيح وبعض الإلكترونات داخله لإظهار النصوص وتحريك المؤشر! تُربَط عادةً باستخدام كبل تسلسلي إلى حاسوب كبير. كان هنالك العديد من الطرفيات التي تنتمي إلى أصناف تجارية مختلفة التي يمتلك كل نوع منها ميزات خاصة للشاشة ولوحة المفاتيح. ولأن جميع الطرفيات تستطيع فهم ASCII على الأقل؛ فكان المبرمجون الذي يطمحون إلى كتابة برمجيات محمولة (أي تعمل على عدّة منصات) يكتبون برامجهم مستخدمين "القاسم المشترك الأصغر" لمزايا الطرفيات. لدى أنظمة يونكس طرق مدروسة للتعامل مع الطرفيات وميزات العرض الخاصة بها. ولأن مطوري مكتبة Readline لم يكونوا واثقين من اسم زر التحكم الإضافي فاخترعوا واحدًا

> واسموه Meta. وعلى الرغم من أن الزر Alt يعمل عمل الزر Meta في لوحات المفاتيح الحديثة؛ إلا أنك تستطيع الضغط على زر Esc الذي يقوم بنفس التأثير عند الضغط مع التعليق (hold) على الزر Alt في حال ما زلت تستخدم الطرفية (يمكنك القيام بذلك في لينُكس!)

الإكمال التلقائي

آلية أُخرى تساعدنا الصدفة فيها تسمى "الإكمال التلقائي"، يحدث الإكمال التلقائي عندما تضغط على زر tab أثناء كتابتك لأمرٍ ما. لنرَ الآن كيف تعمل. لنفترض أن مجلد المنزل لديك يحتوي على:

```
[me@linuxbox ~]$ ls
Desktop       ls-output.txt    Pictures     Templates        Videos
Documents     Music            Public
```

جرب الآن طباعة ما يلي لكن **لا تضغط على زر Enter:**

```
[me@linuxbox ~]$ ls l
```

اضغط الآن على زر tab:

```
[me@linuxbox ~]$ ls ls-output.txt
```

هل لاحظت كيف تُكمِل الصدفة الأمر؟ لنجرب مرّة أُخرى. لكن لا تضغط هذه المرّة أيضًا على زر Enter:

```
[me@linuxbox ~]$ ls D
```

ومن ثم اضغط على الزر tab:

```
[me@linuxbox ~]$ ls D
```

لم يحدث أي شيء! فقط زمرّ الجهاز. حدث ذلك لأن "D" يُطابق أكثر من قيد في المجلد. يجب أن يكون هناك دليلٌ كافٍ للصدفة لتحديد القيد:

```
[me@linuxbox ~]$ ls Do
```

ومن ثم اضغط على زر tab:

```
[me@linuxbox ~]$ ls Documents
```

تمت عملية الإكمال بنجاح.

وعلى الرغم من أن المثال السابق قد بيّن الإكمال التلقائي لأسماء الملفات (الذي يعتبر أشهر استخدام للإكمال التلقائي)؛ لكن الإكمال التلقائي يعمل أيضًا على المتغيرات (إذا كان المحرف "$" في أول الكلمة) أو أسماء المستخدمين (إذا بدأت الكلمة برمز "~") أو الأوامر (في أول السطر) بالإضافة إلى أسماء المضيفين (hosts إذا بدأت الكلمة برمز "@"). الإكمال التلقائي لأسماء المضيفين لا يعمل إلا لأسماء المضيفين الموجودين في ملف /etc/hosts.

يوجد عدد من اختصارات المفاتيح التي تُستخدم في الإكمال التلقائي:

الجدول 8-4: أوامر الإكمال

الزر	الشرح
`Alt-?`	إظهار قائمة بالخيارات المتاحة للإكمال. يمكنك القيام بذلك أيضًا بطريقة أسهل في أغلب التوزيعات بالضغط مرّة أُخرى على الزر tab.
`Alt-*`	تضمين جميع خيارات الإكمال الممكنة، قد تستفيد من هذا الاختصار إذا أردت استخدام أكثر من مطابقة واحدة.

يوجد العديد من الاختصارات الأخرى التي أجدها غامضة. يمكنك مشاهدة القائمة الكاملة في صفحة الدليل للصدفة bash تحت قسم "READLINE".

الإكمال التلقائي القابل للبرمجة

تحتوي الإصدارات الأخيرة من bash على خاصية تسمى "الإكمال التلقائي القابل للبرمجة". يسمح الإكمال التلقائي القابل للبرمجة لك (أو باحتمال أكبر، لمطور توزيعتك) بإضافة قواعد إضافية للإكمال التلقائي. تُستخدم هذه الميزة عادةً لإضافة الدعم لبرامج محددة. على سبيل المثال، يمكن إضافة الإكمال التلقائي لخيارات الأوامر أو للسماح بالإكمال لنوع معين من الملفات. تحتوي توزيعة أوبنتو على عدد كبير منها افتراضيًا. يُبنى الإكمال التلقائي القابل للبرمجة بدوال الشِل، التي تمثل سكربت شِل صغير سنشرح طريقة إنشائه في فصول لاحقة. إذا كنت مهتمًا بذلك، فنفذ الأمر:

```
set | less
```

وتأكد فيما إن كانت توزيعتك تدعمهم؛ لا تحتوي جميع التوزيعات عليهم افتراضيًا.

استخدام تأريخ الأوامر

كما اكتشفنا في الفصل الأول، تدير bash قائمةً بالأوامر التي أدخلناها من قبل. يُحتَفَظ بتلك القائمة في ملف في مجلد المنزل الخاص بك يُدعى ".bash_history". تفيد خاصية التأريخ بتقليل الطباعة التي تقوم بها وخصوصًا عندما تستخدمها مع تعديلات سطر الأوامر.

البحث في التأريخ

يُمكنك عرض محتويات قائمة التأريخ في أيّ وقت باستخدام الأمر:

```
[me@linuxbox ~]$ history | less
```

تُخزِّن bash افتراضيًّا آخر خمسمائة أمر تم إدخالهم. سنتعلم طريقة تعديل تلك القيمة في فصل لاحق. لنفترض أننا نريد أن نعرف الأوامر التي استخدمناها لعرض محتوى المجلد usr/bin/. إحدى الطرق هي:

```
[me@linuxbox ~]$ history | grep /usr/bin
```

ولنفترض أننا حصلنا على سطر يحتوي على أمر مثير للاهتمام كالآتي:

```
88 ls -l /usr/bin > ls-output.txt
```

يُمثل الرقم "88" رقم سطر الأمر الظاهر في قائمة التأريخ. بإمكاننا استخدامه فورًا عن طريق نوع آخر من التوسعات يُسمى توسعة التأريخ. لاستخدام الأمر المُكتشَف، نطبع:

```
[me@linuxbox ~]$ !88
```

توسّع bash العبارة "88!" إلى محتويات السطر الثامن والثمانون في قائمة التأريخ. هنالك أشكال أُخرى للتوسعات سنشرحها لاحقًا في هذا الفصل.

توفر bash أيضًا إمكانية البحث في قائمة التأريخ "تفاعليًّا". أي أننا سنخبر bash بأن تبحث في قائمة التأريخ بينما نقوم نحن بطباعة الحروف، سيزيد كل حرف ندخله من قابلية المطابقة للنص الذي نبحث عنه. اضغط على Ctrl-r لبدء البحث العكسي التفاعلي ثم أدخل النص الذي تريد البحث عنه؛ عندما تجده، اضغط على زر Enter لتنفيذ الأمر أو Ctrl-j لنسخ الأمر من قائمة التأريخ إلى سطر الأوامر. للبحث عن المطابقة التالية للنص (البحث نحو "الأعلى") نضغط على Ctrl-r مرة أُخرى. لإنهاء البحث نضغط على Ctrl-g أو Ctrl-c. لنجربها:

```
[me@linuxbox ~]$
```

أولًا، اضغط على `Ctrl-r`:

```
(reverse-i-search)`':
```

يُشير المِحث إلى أننا سنقوم بإجراء بحث عكسي. (كلمة "عكسي" تعني أننا نبحث من "الآن" إلى وقت ما في الماضي). ومن ثم سنبدأ بطباعة نص البحث. في هذا المثال "`/usr/bin`":

```
(reverse-i-search)`/usr/bin': ls -l /usr/bin > ls-output.txt
```

يُظهر البحث النتيجة فوريًّا. يمكننا الآن تنفيذ النتيجة بعد الحصول عليها بالضغط على Enter، أو نسخها إلى سطر الأوامر لتعديلها بالضغط على `Ctrl-j`. لننسخها:

```
[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt
```

يعود إلينا مِحث الصدفة مذخرًا وجاهزًا للانطلاق!

يحتوي الجدول الآتي على قائمة باختصارات لوحة المفاتيح التي تُستخدم مع قائمة التأريخ:

الجدول 8-5: أوامر التأريخ

الزر	الشرح
`Ctrl-p`	الانتقال إلى قيد التأريخ السابق؛ كالضغط على السهم العلوي.
`Ctrl-n`	الانتقال إلى قيد التأريخ التالي؛ كالضغط على السهم السفلي.
`Alt-<`	الانتقال إلى بداية (أعلى) قائمة التأريخ.
`Alt->`	الانتقال إلى نهاية (أسفل) قائمة التأريخ. أي الأمر الحالي.
`Ctrl-r`	بحث عكسي تفاعلي. البحث بشكل تزايدي من الأمر الحالي إلى أعلى القائمة.
`Alt-p`	بدء البحث العكسي غير التفاعلي. تقوم في هذا النوع بكتابة نص البحث وتضغط على رز Enter قبل أن يتم البحث.
`Alt-n`	بدء بحث أمامي (من الأعلى إلى الأسفل، أو من القديم إلى الجديد) غير تفاعلي.
`Ctrl-o`	تنفيذ الأمر الحالي في قائمة التأريخ ومن ثم الانتقال إلى الأمر الذي يليه. يُفيد هذا الاختصار إذا أردت إعادة تنفيذ سلسلة من الأوامر في قائمة التأريخ.

توسيع تأريخ الأوامر

توفر الصدفة نوعًا خاصًا من التوسعة يُستخدم للقيود (جمع قيد أي سجل) في قائمة التأريخ باستخدام الرمز "!". شاهدنا سابقًا استخدام علامة التعجب يتبعها رقم، لإدراج قيد من قائمة التأريخ. هنالك عدد من التوسعات الأخرى:

الجدول 8-6: أوامر توسعة التأريخ

التعبير	الشرح
`!!`	إعادة تنفيذ آخر أمر. ربما يكون من الأسهل الضغط على السهم العلوي ومن ثم `Enter`.
`!number`	إعادة تنفيذ الأمر الذي يكون ترتيبه في قائمة التأريخ مساويًا للرقم "`number`".
`!string`	تنفيذ آخر أمر في قائمة التأريخ يبدأ بالعبارة "`string`".
`!?string`	تنفيذ آخر أمر في قائمة التأريخ يحتوي العبارة "`string`".

أحذرك من استخدام التعبيرَين "`!string`" و "`!?string`" إلا إذا كنت متأكدًا تمامًا من محتويات قائمة التأريخ لديك.

هنالك العديد من الخيارات المتوفر لعمليات توسعة التأريخ، لكن هذا الموضوع أخذ أكثر من ما يستحق وربما ستنفجر رؤوسنا إذا أكملنا الشرح! يحتوي قسم "`HISTORY EXPANSION`" في صفحة الدليل bash على تفاصيل أكثر. أتركك لتكتشفها!

الأمر `script`

بالإضافة إلى خاصية التأريخ في bash،توفر أغلب توزيعات لينُكس برنامجًا يُسمى `script`، يُستخدم لتسجيل كامل جلسة الصدفة ويُخزّنها إلى ملف. الشكل العام لاستخدمه هو:

```
script [file]
```

حيث `file` هو اسم الملف الذي سيُستخدم لتخزين محتوى الجلسة. سيُستخدم الملف `typescript` إذا لم يُحدد اسم ملف التسجيل. راجع صفحة الدليل للأمر `script` للحصول على معلومات مفصّلة عن خياراته وميزاته.

الخلاصة

لقد شرحنا في هذا الفصل عددًا من "خدع" لوحة المفاتيح التي توفرها الصدفة لتقليل عبء العمل على

المستخدمين. أظن أنك ستعود إلى هذا الفصل وتتعلم عددًا من اختصارات لوحة المفاتيح مع مرور الوقت وعند زيادة تعاملك مع سطر الأوامر. لكن حاليًا اعتبرهم ميزات اختيارية، لكنها مُفيدة جدًا.

الفصل التاسع:

الأذونات

تختلف أنظمة التشغيل التي تتبع عرف يونكس عن تلك التي تتبع عُرف MS-DOS ليس بأنها متعددة المهام فحسب، ولكنها أنظمة متعددة المستخدمين أيضًا.

ماذا يعني هذا الكلام بدقة؟ يعني أنه بالإمكان استخدام النظام من أكثر من مستخدم في آن واحد. وعلى الرغم من أن الحاسوب العادي قد لا يحتوي على أكثر من لوحة مفاتيح وشاشة واحدة فقط، إلا أنه قابل للاستخدام من أكثر من شخص. على سبيل المثال، يستطيع المستخدمون "البعيدون" الدخول إلى الحاسوب عبر ssh (الصدفة الآمنة [secure shell]) إذا كان الحاسوب موصولًا إلى شبكة ما أو إلى الإنترنت. في الواقع، يستطيع المستخدمون عن بعد تنفيذ البرمجيات التي تعتمد على الواجهة الرسومية وإظهار النتائج على شاشتهم عن بعد! يدعم مدير العرض X ذلك في بنيته وتصميمه الأساسي.

ليست قدرة نظام لينُكس على تعدد المستخدمين وليدة الساعة، وإنما هي ميزة أساسية مدمجة دمجًا عميقًا في تصميم النظام. سيكون الأمر منطقيًا للغاية إذا أخذت بعين الاعتبار البيئة التي نشأ فيها نظام يونكس. منذ العديد من السنوات، كانت الحواسيب كبيرة وغالية الثمن ومركزية قبل أن تكون "شخصية". كان حاسوب الجامعة العادي مكوَّن من حاسوب مركزي كبير موجود في أحد المباني، وطرفيات متوزعة في حرم الجامعة، تتصل جميعها بالحاسوب المركزي. كان الحاسوب يدعم العديد من المستخدمين في آن واحد.

أُنشئت طريقة لحماية المستخدمين من بعضهم البعض لكي يكون استخدام الحاسوب أمرًا عمليًا. لذا، فإن أفعال أي مستخدم لا تسمح له أن يُعطِب الحاسوب أو أن يتعامل مع ملفات يملكها مستخدم آخر.

سنلقي في هذا الفصل نظرةً على جزء مهم من أمن النظام وسنتعرف على الأوامر الآتية:

- id – إظهار هوية المستخدم.
- chmod – تغيير أذونات ملفٍ ما.
- umask – تحديد الأذونات الابتدائية الافتراضية.
- su – تشغيل صدفة كمستخدم آخر.
- sudo – تنفيذ أمر ما كمستخدم آخر.
- chown – تغيير مالك الملف.
- chgrp – تغيير المجموعة المالكة للملف.

• passwd – تغيير كلمة مرور المستخدم.

المالكون وأعضاء المجموعة وأي شخص آخر

ربما واجهتنا مشكلة عند محاولتنا عرض محتوى بعض الملفات كالملف etc/shadow/ عندما كُنّا نستكشف النظام في الفصل الثالث:

```
[me@linuxbox ~]$ file /etc/shadow
/etc/shadow: regular file, no read permission
[me@linuxbox ~]$ less /etc/shadow
/etc/shadow: Permission denied
```

سبب رسالة الخطأ السابق هو أننا، كمستخدمين عاديين، لا نملك امتيازات كافية لقراءة هذا الملف.

في نموذج الحماية الذي يستخدمه يونكس. بإمكان المستخدم أن يملك الملفات والمجلدات. عندما يملك المستخدم ملفًا أو مجلدًا، فإنه يتحكم في أذونات الوصول إليه. على الكفة الأخرى، يستطيع المستخدمون الانتماء إلى "مجموعة" تحتوي مستخدمًا واحدًا أو أكثر، ويمكن إعطاؤهم امتيازات للوصول إلى الملفات والمجلدات من قِبل مالكيها. بالإضافة إلى تحديد الأذونات للمجموعة، يستطيع المالك إعطاء بعض الأذونات إلى أي مستخدم آخر في النظام، الذي يُشار إليه بمصطلحات يونكس بـ"العالم". استخدم الأمر id للحصول على معلومات حول هويتك:

```
[me@linuxbox ~]$ id
uid=500(me) gid=500(me) groups=500(me)
```

لنلقِ نظرة على المخرجات. عندما تُنشَأ حسابات المستخدمين، فسيُسند رقم يُسمى user ID أو uid ومن ثم يُربَط باسم المستخدم. ويُسند للمستخدم أيضًا رقم المجموعة الرئيسية primary group ID أو gid ويمكن أن يكون المستخدم عضوًا في مجموعات أُخرى. أُخِذَ ناتج المثال السابق من توزيعة فيدورا. لكن قد يختلف الناتج قليلًا في بعض التوزيعات الأُخرى، كتوزيعة أوبنتو:

```
[me@linuxbox ~]$ id
uid=1000(me) gid=1000(me)
groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(vid
eo),46(plugdev),108(lpadmin),114(admin),1000(me)
```

كما لحظنا؛ تختلف أرقام uid و gid. لأن توزيعة فيدورا تبدأ أرقام المستخدمين العاديين من الرقم 500، بينما تبدأه أوبنتو من 1000. ويمكننا أيضًا ملاحظة أن مستخدم أوبنتو ينتمي إلى عددٍ أكبر بكثير من المجموعات

بالمقارنة مع مستخدم فيدورا. سبب ذلك هو طريقة إدارة أوبنتو لامتيازات أقراص وخدمات النظام.

من أين تأتي هذه المعلومات؟ كما في العديد من الأشياء في لينُكس، فهي تأتي من ملفَين نصيّين. تُعرَّف حسابات المستخدمين في ملف /etc/passwd، وتُعرَّف المجموعات في ملف /etc/group. يُعدَّل هذان الملفان بالإضافة إلى ملف /etc/shadow الذي يحتوي على معلومات حول كلمة المرور الخاصة بالمستخدم عندما تُنشأ حسابات المستخدمين أو المجموعات. ولكل حساب مستخدم، فإن الملف /etc/passwd يحتوي على اسم المستخدم (الخاص بالدخول) و uid و gid واسم المستخدم الحقيقي ومسار مجلد المنزل والصدفة الافتراضية. إذا دققتَ في محتوى الملفين /etc/passwd و /etc/group فسوف تلاحظ وجود حسابات لمستخدمين آخرين عدا المستخدمين العاديين، حيث يوجد حسابات للمستخدم الجذر (تكون قيمة uid للمستخدم الجذر تساوي القيمة 0)، بالإضافة إلى مستخدمي النظام الآخرين.

في الفصل القادم سنشرح "العمليات"، وسوف تشاهد أن بعض هؤلاء "المستخدمين" مشغول جدًا.

وعلى الرغم من أن الأنظمة الشبيهة بِيونكس تُضيف المستخدمين العاديين إلى مجموعة شائعة تدعى "users"، إلا أن أنظمة لينُكس الحديثة تُنشئ مجموعةً فريدةً تحتوي مستخدمًا واحدًا بنفس اسم المستخدم. مما يُسهِّل إدارة بعض أنواع الأذونات.

القراءة والكتابة والتنفيذ

تُعرّف أذونات الوصول إلى الملفات والمجلدات بثلاثة شروط هي إذن القراءة وإذن الكتابة وإذن التنفيذ (read, write, execute). نستطيع معرفة طريقة عمل تلك الأذونات إذا ألقينا نظرة على ناتج الأمر ls:

```
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1    me    me  0   2008-03-06   14:52   foo.txt
```

أول عشرة محارف في المثال السابق تسمى خاصيات الملف. أول تلك المحارف هو "نوع الملف". يحتوي الجدول الآتي على أبرز الأنواع (بالإضافة إلى الأنواع غير الشائعة أيضًا!) التي يمكن أن تواجهها:

الجدول 9-1: أنواع الملفات

الخاصية	نوع الملف
-	ملف عادي.
d	مجلد.
l	وصلة رمزية. لاحظ أن باقي خاصيات الملف تكون دائمًا "rwxrwxrwx" وهي ليست الخاصيات

	الحقيقية للملف الذي تُشير إليه الوصلة.
c	ملف محرفي خاص (character special file). يُشير هذا النوع من الملفات إلى الأجهزة التي تتعامل مع البيانات كمجرى من البايتات (stream of bytes) كالطرفية أو المودم.
b	ملف كتلي خاص (block special file). يُشير هذا النوع من الملفات إلى الجهاز الذي يتعامل مع البيانات ككتل، كالقرص الصلب أو CD-ROM.

المحارف التسعة الباقية تدعى "نمط الملف" (file mode) والتي تُمثل أذونات القراءة والكتابة والتنفيذ لمالك الملف وللمجموعة المالكة له ولأي مستخدم آخر:

العالم	المجموعة	المالك
rwx	rwx	rwx

لخاصيات القراءة r والكتابة w والتنفيذ x المعاني الآتية على الملفات والمجلدات:

الجدول 9-2: الأذونات

الخاصية	الملفات	المجلدات
r	تسمح للملفات بأن تُفتح وتُقرأ.	تسمح بعرض محتويات مجلد (عرض قائمة بالملفات والمجلدات التي تحويه) وذلك إذا حُددت خاصية التنفيذ.
w	تسمح بالكتابة على الملف أو حذف جميع محتوياته، لكن هذه الخاصية لا تسمح بنقل أو إعادة تسمية الملف. القدرة على نقل وإعادة تسمية الملفات تتعلق بخاصيات المجلد.	السماح بإنشاء وحذف وإعادة تسمية الملفات داخل المجلد وذلك في حال حُدِّدَت خاصية التنفيذ.
x	تسمح باعتبار الملف برنامجًا قابلًا للتنفيذ. يجب أن تُحَدَّد خاصية القراءة للبرمجيات المكتوبة في إحدى لغات السكربتات؛ كي تستطيع تنفيذها.	السماح بدخول المجلد (أي تنفيذ `cd directory`).

وهذه بعض الأمثلة عن خاصيات الملفات ومعانيها:

الجدول 9-3: أمثلة عن الأذونات

الخاصية	نوع الملف
`-rwx------`	ملف عادي قابل للقراءة والكتابة والتنفيذ من قِبل مالك الملف. لا يملك أي مستخدم آخر أيّة أذونات.
`-rw-------`	ملف عادي قابل للقراءة والكتابة من قِبل مالك الملف. لا يملك أي مستخدم آخر أيّة أذونات.
`-rw-r--r--`	ملف عادي قابل للقراءة والكتابة من قِبل مالك الملف. يمكن لأعضاء المجموعة المالكة ولباقي المستخدمين قراءة الملف.
`-rwxr-xr-x`	ملف عادي قابل للقراءة والكتابة والتنفيذ من قِبل المستخدم المالك. وقابل للقراءة والتنفيذ من قِبل أعضاء المجموعة وأي شخص آخر.
`-rw-rw----`	ملف عادي قابل للقراءة والكتابة من قِبل مالك المستخدم والمجموعة المالكة فقط.
`lrwxrwxrwx`	وصلة رمزية. جميع أذونات الوصلات الرمزية "زائفة". الأذونات الحقيقة تبقى مرتبطةً بالملف الأصلي الذي تُشير إليه الوصلة.
`drwxrwx---`	مجلد يمكن للمالك ولأعضاء المجموعة الدخول وإنشاء وإعادة تسمية وحذف الملفات داخله.
`drwxr-x---`	مجلد يمكن للمالك الدخول وإنشاء وإعادة تسمية وحذف الملفات داخله. ويمكن لأعضاء المجموعة المالكة الدخول ولكن ليس إنشاء أو حذف أو إعادة تسمية الملفات.

تغيير أذونات الملف باستخدام chmod

لتغيير نمط (mode عادةً ما يُشار إليه بالأذونات) ملف أو مجلد ما، فإننا نستخدم الأمر chmod. يجدر بالذكر أن مالك الملف أو المستخدم الجذر هما الوحيدان اللذان يستطيعان تغيير أذونات ملف أو مجلد. يدعم الأمر chmod طريقتَين لتحديد الأذونات: التمثيل باستخدام الأرقام في النظام الثماني، أو التمثيل الرمزي (أي تمثيل الأذونات على شكل حروف). سنشرح التمثيل باستخدام الأرقام أولًا.

ما هو نظام العد الثماني؟

نظـام العـد الثمـاني (`Octal` الـذي يعتمـد علـى الأسـاس 8) وابـن عمـه نظـام العـد السـت عشـري (hexadecimal الـذي يعتمـد علـى الأسـاس 16) همـا نظامـا عـدّ يُسـتخدمان للتعبيـر عـن الأعـداد فـي الحواسيب. ولأننا كبشر وُلدنا (أو على الأقل، وُلِدَ أغلبنا) بعشرة أصابع، فإننا نعّد متعمدين علـى الأسـاس 10. في الكفة المقابلة، ولِدت الحواسيب بإصبع واحد. لذا، فإن جميع عمليات العد التي تقوم بها تعتمد على نظام العد الثنائي (binary) وتعد كالآتي:

```
0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011...
```

يكون العد في النظام الثماني بالأعداد من الصفر حتى السبعة، كالآتي:

```
0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21...
```

أما النظام الست عشري فيستخدم الأرقام من صفر إلى تسعة بالإضافة إلى الأحرف من الحرف "A" إلى الحرف "F":

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13...
```

ربما نجد بعض المنطق في استخدام النظام الثنائي، لكن ما الفائدة المرجوة من نظامي العد الثماني والست عشري؟ الجواب يكمن في سهولة قراءة الأعداد المُمثلة بتلك الأنظمة. غالبًا ما تُمثَّل البيانات علـى شـكل "متسلسـلة ثنائيـة". علـى سـبيل المثـال لـون RGB، كـل بكسـل يُمثِّـل، فـي أغلـب شاشـات الحاسوب حاليًا، بثلاثة مكونات للألوان: ثمانيـة بتـات للـون الأحمـر R، وثمانيـة بتـات للـون الأخضـر G، وثمانية بتات للون الأزرق B. يُمثَّل اللون الأزرق بأربعٍ وعشرين رقمًا:

010000110110111111001101

كيف ستتمكن من قراءة وكتابة هذا النوع من الأرقام طوال اليوم؟ لا أظن بأنك ستفعل ذلك! يفيد نظام أعداد آخر هنا. كل رقم (أو حرف) في نظام العد الست عشري يُمثِّل أربعة بتات. وكل رقم في نظام العد الثماني يُمثِّل ثلاثة بتات. لذا فإن اللون الأزرق ذا الأربعٍ والعشرين رقمًا يمكن تمثيله بنظام الست عشري بستة أرقام:

436FCD

ولأن الأرقام في النظام الست عشري "تتوافق" مع البتات في النظام الثنائي، فتستطيع معرفة أن قيمة اللون الأحمر هي "43" واللون الأخضر "6F" واللون الأزرق "CD".

في هذه الأيام، يُستخدم النظام الست عشري (عادةً ما يُشار إليه بالكلمة hex) أكثر من النظام الثماني. إلا أننا سنرى أن إمكانية النظام الثماني لتمثيل ثلاثة بتات ستفيدنا للغاية...

نستخدم الأرقام للتعبير عن الأذونات في الطريقة التي تعتمد على النظام الثماني. ولأن كل رقم في النظام الثماني يُمثِّل ثلاثة بتات؛ فسيُستخدم مع أذونات الملف. يُزيل الجدول الآتي الغموض عن كلامنا السابق:

الجدول 9-4: أذونات الملف بالنظامين الثنائي والثماني

النظام الثماني	النظام الثنائي	الإذن
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

تستطيع تحديد أذونات الملف للمستخدم المالك وللمجموعة المالكة ولباقي المستخدمين باستخدام ثلاثة أرقام تعتمد على نظام العد الثماني:

```
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 me me   0   2008-03-06  14:52  foo.txt
[me@linuxbox ~]$ chmod 600 foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw------- 1 me me   0   2008-03-06  14:52  foo.txt
```

تمكّنا من إعطاء أذونات القراءة والكتابة للمالك بتمرير الخيار "600"، وأزلنا جميع الأذونات لباقي المستخدمين (بما فيهم أعضاء المجموعة المالكة). ربما يكون تذكر أرقام الأذونات أمرًا متعبًا، إلا أنك لا تستخدم عادةً إلا القيم الشهيرة الآتية: 7 (rwx) و 6 (-rw) و 5 (r-x) و 4 (--r) و 0 (---).

يدعم الأمر chmod تحديد الأذونات بالتمثيل الرمزي. يُقسَّم التمثيل الرمزي إلى ثلاثة أقسام: من ستُغيّر الأذونات له (المالك أو المجموعة أو المستخدمين الآخرين)، والعملية التي ستُنفَّذ، والأذونات التي ستُحَدَّد. يُستخدم دمج بين أربعة محارف لتحديد من ستُغيَر الأذونات له هي "u" و "g" و "o" و "a" التي تعني الآتي:

الجدول 9-5: معاني رموز الأمر chmod

الرمز	الشرح
u	اختصار للكلمة "user" أي تعني مالك الملف أو المجلد.
g	المجموعة المالكة.
o	اختصار لكملة "others" التي تعني المستخدمين الآخرين.
a	اختصار للكلمة "all" أي تعني دمج ما بين "u" و "g" و "o".

سيُستخدم "all" افتراضيًّا في حال لم يُحدد أيٌّ من المحارف السابقة. العملية التي يمكن تنفيذها تكون إما "+" التي تعني إضافة الأذونات، أو "-" أي نزع الأذونات، أو "=" التي تعني ضبط الأذونات المحددة وإزالة أيّة أذونات أُخرى.

تُحَدَّد الأذونات بالأحرف "r" و "w" و "x". يحتوي الجدول الآتي على بعض الأمثلة عن استخدام التمثيل الرمزي:

الجدول 9-6: أمثلة عن التمثيل الرمزي في الأمر chmod

الرمز	الشرح
u+x	إضافة إذن التنفيذ للمالك.
u-x	نزع إذن التنفيذ من المالك.
+x	إضافة إذن التنفيذ للمالك وللمجموعة المالكة ولباقي المستخدمين. يقوم بنفس تأثير a+x.
o-rw	إزالة أذونات القراءة والكتابة من أي مستخدم عدا المالك والمجموعة المالكة.
go=rw	تحديد إذن القراءة والكتابة لأعضاء المجموعة المالكة وأي مستخدم آخر. إذا كان للمجموعة أو لباقي المستخدمين إذن التنفيذ، فسيُزال.
u+x,go=rx	إضافة إذن التنفيذ للمالك وتحديد إذن المجموعة وباقي المستخدمين إلى القراءة والتنفيذ. يمكن فصل أكثر من عملية ضبط في نفس الأمر بفاصلة ",".

يُفضِّل بعض الأشخاص استخدام الطريقة الأولى التي تعتمد على النظام الثماني، بينما يُفضِّل البعض الآخر طريقة التمثيل الرمزي. توفر الطريقة الثانية إمكانية تغيير إذن واحد دون أن تُمَسّ باقي الأذونات.

ألقِ نظرة على صفحة الدليل للأمر chmod لتفاصيل كاملة وقائمة بالخيارات الممكنة. لكن كن حذرًا من الخيار "‎--recursive" لأنه يعمل على الملفات والمجلدات، من النادر جدًا أن نريد أن تكون للمجلدات والملفات نفس الأذونات.

تحديد أذونات الملفات باستخدام الواجهة الرسومية

الآن وبعد أن تعرفنا على آلية ضبط أذونات الملفات والمجلدات، حان الوقت لفهم مربعات الحوار الموجودة في الواجهة الرسومية. تستطيع أن تحصل على مربع حوار الخصائص في كلي مديري الملفات نوتاليز (غنوم) ودولفين (كدي) بالضغط بالزر الأيمن والنقر فوق "خصائص" (properties). الصورة الآتية من واجهة كدي:



الشكل 2: مربع حوار الخصائص في واجهة كدي

تستطيع هنا مشاهدة الأذونات للمالك وللمجموعة المالكة وباقي المستخدمين. سيؤدي الضغط على زر "أذون متقدمة" (Advanced Permissions) في كدي إلى إظهار مربع حوار آخر يُمكّنك من تحديد كل إذن على حدة.

umask: تحديد الصلاحيات الافتراضية

يتحكم الأمر umask بالأذونات الافتراضية التي تعطى إلى ملفٍ ما عند إنشائه. يستخدم umask النظام الثماني لكي يُحدد "قناع" البتات التي ستُزال من أذونات الملفات. لنلقِ نظرةً عليه:

```
[me@linuxbox ~]$ rm -f foo.txt
[me@linuxbox ~]$ umask
0002
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1   me   me   0   2008-03-06   14:53   foo.txt
```

في البداية، حذفنا النسخة القديمة من ملف `foo.txt` لنتأكد من أننا سننشئ ملفًا جديدًا. ومن ثم نفذنا الأمر umask بدون أيّة وسائط لكي نرى القيمة الحالية. أخرج الأمر القيمة 0002 (القيمة 0022 هي قيمة شهيرة أيضًا) التي تُعبّر عن التمثيل الثماني لقناع الأذونات. ومن ثم أنشأنا نسخةً جديدةً من الملف `foo.txt` وعاينا الأذونات المُعطاة له.

كما لاحظت، لدى المستخدم المالك والمجموعة المالكة أذونات القراءة والكتابة. بالإضافة إلى أذونات القراءة فقط لباقي المستخدمين. سبب عدم امتلاك باقي المستخدمين لأذونات الكتابة هو قيمة "القناع". لنعد تنفيذ المثال السابق لكن هذه المرّة سنُحدد قيمة القناع:

```
[me@linuxbox ~]$ rm foo.txt
[me@linuxbox ~]$ umask 0000
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-rw-   1   me   me   0   2008-03-06   14:58   foo.txt
```

عندما ضبطنا قيمة القناع إلى القيمة 0000 (أي أوقفنا عمل القناع) نستطيع ملاحظة أن الملف أصبح قابلًا للكتابة من أيّ مستخدم. ولكي نفهم آلية عمل القناع، سنلقي نظرة أُخرى على نظام العد الثماني. إذا حولنا قيمة القناع إلى النظام الثنائي، وقارناه بأذونات الملف فسوف نستطيع معرفة ما الذي جرى:

أذونات الملف الافتراضية:	`--- rw- rw- rw-`
القناع:	`000 000 000 010`
النتيجة:	`--- rw- rw- r--`

تجاهل الأصفار الموجودة في البداية (سنتحدث عنهم بعد دقيقة) ولاحظ كيف سيُزال الإذن المقابل لكل رقم 1 في القناع. وفي حالتنا هذه هو إذن الكتابة لباقي المستخدمين. آلية عمل القناع هي كالآتي: عند كل رقم "1" يظهر في القيمة الثنائية للقناع؛ يُزال الإذن الموافق له. في هذه الحالة، إذن الكتابة لباقي المستخدمين. لننظر الآن إلى الذي يفعله القناع ذو القيمة 0022:

أذونات الملف الافتراضية:	`--- rw- rw- rw-`
القناع:	`000 000 010 010`
النتيجة:	`--- rw- r-- r--`

مرة أُخرى، أُزيلت الأذونات الموافقة لأي رقم "1". جرب بعض القيم (بما فيها الرقم 7) لكي تستوعب كيف يُستخدم umask. لا تنسَ أن تعيد القيمة الافتراضية عند الانتهاء من التجارب:

```
[me@linuxbox ~]$ rm foo.txt; umask 0002
```

لا تحتاج إلى تغيير القناع في أغلب الأحيان؛ القيمة الافتراضية لتوزيعتك لا بأس بها. لكن قد تحتاج إلى التحكم في قيمتها في بعض الأنظمة عالية الحماية.

بعض الأذونات الخاصة

على الرغم من أننا نشاهد الأذونات المكتوبة بنظام الأرقام الثماني على شكل ثلاثة أرقام، إلا أنه أصح تقنيًا أن تُمثَّل بواسطة أربعة أرقام. لماذا؟ لوجود أذونات أُخرى عدا أذونات القراءة والكتابة والتنفيذ لكنها أقل استخدامًا.

أولى تلك الخاصيات تُدعى setuid bit (تُمثَّل في النظام الثماني على شكل 4000). عندما تُطبَّق على ملف تنفيذي، فسيتم "تغيير" معرّف المستخدم الذي يُشَغِّل البرنامج إلى معرّف المستخدم المالك له. يُعطى هذا الإذن عادةً لبعض البرامج التي يملكها المستخدم الجذر. عندما يُشغِّل مستخدم عادي برنامجًا له إذن "setuid root" فسيُنفّذ البرنامج بصلاحيات الجذر. هذا يسمح للمستخدم العادي بالوصول إلى ملفات ومجلدات لا يملك إذن الوصول إليها. وهذا ما قد يشكل نقاط ضعف في حماية النظام، لذا، من الضروري التقليل من عدد الملفات التنفيذية التي لها هذا الإذن.

النوع الثاني من الأذونات الخاصة يُسمى setgid bit (تُمثل في النظام الثماني على شكل 2000). إذا أُعطي هذا الإذن لمجلدٍ ما، فستتحول ملكية جميع الملفات في هذا المجلد إلى المجموعة المالكة للمجلد عوضًا عن المجموعة الافتراضية للمستخدم.

النوع الثالث يدعى `sticky bit` (1000 في النظام الثماني). في يونكس، كان من الممكن تحديد هذا الإذن لكي لا يُسمَح "بتبديل" الملفات التنفيذية. يتجاهل لينُكس ذاك النوع من الأذونات إذا حُدِّدَ على ملف. لكن إذا طُبِّق على مجلد فإنه يمنع المستخدمين من حذف أو إعادة تسمية الملفات إلا إذا كان المستخدم مالكًا للمجلد أو مالكًا للملف، أو أنه المستخدم الجذر. يُستخدم هذا الإذن عادةً للتحكم في

الوصول إلى مجلد مشترك، كالمجلد tmp/.

هذه بعض الأمثلة عن استخدام الأمر chmod مع التمثيل الرمزي للأذونات الخاصة السابقة. أولًا إضافة إذن setuid إلى برنامج:

```
chmod u+s program
```

تحديد إذن setgid إلى مجلد:

```
chmod g+s dir
```

أخيرًا، تحديد إذن sticky bit إلى مجلد:

```
chmod +t dir
```

يمكنك معرفة الصلاحيات الخاصة المُطبقة من ناتج الأمر ls. الآتي هو برنامج لديه إذن setuid:

```
-rwsr-xr-x
```

مجلد لديه إذن setgid:

```
drwxrwsr-x
```

مجلد بإذن sticky bit:

```
drwxrwxrwt
```

تغيير الهوية

تحتاج، في بعض الأحيان، إلى تغيير الهوية الخاصة بك إلى هوية مستخدم آخر. يكون السبب غالبًا هو الحاجة إلى الحصول على امتيازات الجذر للقيام بمهمة إدارية. وبالإمكان أيضًا التحويل إلى حساب مستخدم عادي آخر لعدة أغراض كاختبار الحساب على سبيل المثال. توجد ثلاث طرق لتغيير الهوية :

1. تسجيل الخروج ومن ثم تسجيل الدخول بحساب المستخدم الآخر.
2. استخدام الأمر su.
3. استخدام الأمر sudo.

لن نتطرق إلى الطريقة الأولى لأنها مفهومة وغير محبذة بالمقارنة مع الطريقتَين الأُخريتَين. يسمح الأمر su بالحصول على هوية مستخدمٍ آخر ضمن جلسة الصدفة نفسها؛ إما ببدء جلسة جديدة بحساب المستخدم الآخر، أو تنفيذ أمر واحد فقط بذاك الحساب. يسمح الأمر sudo لمدير النظام أن يضبط ملف الإعدادات etc/sudoers/ ويُحَدِّد الأوامر التي يستطيع مستخدمون محددون تنفيذها تحت حساب آخر. اختيار أحد الأمرين السابقين يعتمد على التوزيعة التي تستخدمها. كلا الأمرَين السابقَين موجودٌ في أغلب توزيعات لينُكس، إلا أن بعض التوزيعات تُفضِّل أحد الأمرين على الأمر الآخر. سنبدأ أولًا مع الأمر su.

تشغيل صدفة بحساب مستخدم آخر

يُستخدَم الأمر su لبدء صدفة كمستخدم آخر. شكل الأمر العام:

```
su [-[l]] [user]
```

إذا أُستخدم الخيار "l-"؛ فجلسة الصدفة التي ستُنشأ هي "صدفة الدخول" (login shell) للمستخدم المحدد. ذلك الكلام يعني أنه ستُستخدم بيئة المستخدم الآخر، ويتحول مجلد العمل الحالي إلى مجلد المنزل للمستخدم الآخر. وهذا ما نريد فعله عادةً. إذا لم يُحدد اسم المستخدم فسيُعتبر المستخدم هو المستخدم الجذر. لاحظ أن الخيار "l-" يمكن اختصاره إلى الشكل "-" وهو الشكل الذي يُستخدم عادةً. ننفذ الأمر الآتي لبدء جلسة صدفة بحساب المستخدم الجذر:

```
[me@linuxbox ~]$ su -
Password:
[root@linuxbox ~]#
```

ستُطلب، بعد إدخال الأمر، كلمة مرور حساب المستخدم الجذر. إذا أُدخلت بشكل صحيح، فسيظهر مِحث جديد ينبهنا إلى أن الصدفة لها امتيازات الجذر (باستخدام الرمز "#" بدلًا من "$") ومجلد العمل الحالي هو مجلد المنزل للمستخدم الجذر (غالبًا ما يكون root/). نستطيع الآن أن ننفذ الأوامر بحساب الجذر في جلسة الصدفة الجديدة. نطبع الأمر exit للعودة إلى الصدفة السابقة:

```
[root@linuxbox ~]# exit
[me@linuxbox ~]$
```

ويمكن أيضًا تنفيذ أمر واحد دون إنشاء جلسة تفاعلية جديدة وذلك باستخدام الأمر su على الشكل الآتي:

```
su -c 'command'
```

باستخدام هذا الشكل، سيُمرر الأمر إلى الصدفة الجديدة لكي تنفذها. من المهم أن نضع الأوامر بين علامتي اقتباس مفردتين، حيث لا نريد أن تتم عملية التوسعة في الصدفة الحالية:

```
[me@linuxbox ~]$ su -c 'ls -l /root/*'
Password:
-rw------- 1 root root 754 2007-08-11 03:19 /root/anaconda-ks.cfg
/root/Mail:
total 0
[me@linuxbox ~]$
```

تنفيذ أمر كمستخدم آخر باستخدام sudo

يُشبه الأمر sudo الأمر su بوجوه عديدة. إلا أنه يحتوي على ميزات مهمة إضافية. يضبط مدير النظام عادةً إعدادات sudo للسماح للمستخدم العادي بتنفيذ الأوامر كمستخدم آخر (المستخدم الجذر عادةً) بطريقة مرنة وقابلة للتحكم. يمكن أن يُسمَح للمستخدم بتنفيذ أمر واحد أو أكثر فقط كمستخدم آخر. فرق آخر مهم هو أنه لاستخدام الأمر sudo، فإننا لا نحتاج إلى معرفة كلمة مرور المستخدم الجذر. يطبع المستخدم كلمة المرور الخاصة به كي نتأكد من هويته (الاستيثاق). لنفترض أن الأمر sudo قد تمت تهيئته للسماح لنا باستخدام برنامج لأخذ نسخة احتياطية يدعى "backup_script" الذي يتطلب امتيازات الجذر. فيكون أمر sudo على الشكل الآتي:

```
[me@linuxbox ~]$ sudo backup_script
Password:
System Backup Starting...
```

سئُسأل، بعد طباعة الأمر، عن كلمة المرور الخاصة بنا (وليس كلمة المرور الخاصة بالمستخدم الجذر)، ويُنفّذ الأمر بعد الاستيثاق. فرق مهم بين sudo و su هو أن sudo لا يبدأ جلسة صدفة جديدة، ولا يستورد "البيئة" الخاصة بالمستخدم الآخر. هذا يعني أننا لا نحتاج إلى تضمين الأمر الذي نريد تنفيذه داخل علامتي اقتباس. لكن يمكن أن يُغيّر هذا السلوك باستخدام مختلف الخيارات. راجع صفحة الدليل للأمر sudo للمزيد من المعلومات.

نستخدم الخيار "-l" لعرض الامتيازات التي نملكها عن طريق استخدامنا للأمر sudo:

```
[me@linuxbox ~]$ sudo -l
User me may run the following commands on this host:
    (ALL) ALL
```

أوبنتو و sudo

أحد أبرز المشاكل التي تواجه المستخدمين العاديين هي عند تنفيذ بعض المهام التي تتطلب امتيازات الجذر. تتضمن هذه المهام: تثبيت وتحديث البرامج وتعديل ملفات إعدادات النظام والوصول إلى الأجهزة. يتم ذلك عادةً في عالم ويندوز بإعطاء المستخدم بعض الامتيازات الإدارية، وهذا ما يُمكّن المستخدمين من تنفيذ تلك المهام. لكن ذلك يؤدي أيضًا إلى تشغيل البرامج بنفس الامتيازات. والذي يؤدي إلى أشياء لا تُحمد عقباها. كالبرمجيات الخبيثة والفيروسات التي ستسيطر على الحاسوب

بأكمله!

يوجد عادةً فارق كبير بين المستخدمين العاديين والمدراء في عالم يونكس؛ وذلك بالاستناد إلى آلية تعدد المستخدمين. ولا تُستخدم عادةً امتيازات الجذر إلا وقت الحاجة، وذلك باستخدام الأمريَن su و sudo.

اعتمدت معظم توزيعات لينُكس على su لعدة سنوات، لأن su لا يحتاج إلى الإعداد كما في sudo وكان من الطبيعي وجود حساب "root" في النظام. وهذا ما شكّل مشكلة هي "إجبار" المستخدمين على استخدام حساب الجذر في غير وقت الحاجة. في الحقيقة، يعمل بعض المستخدمين على أنظمتهم بحساب الجذر فقط! وذلك للتخلص من رسائل "permission denied" المزعجة. هذه الفكرة غير السديدة تُخفِّض مدى أمان أنظمة لينُكس إلى مستوى أنظمة ويندوز!

عندما أُنشئت توزيعة أوبنتو، اعتمدت طريقًا آخر ألا وهو إيقاف حساب الجذر افتراضيًّا (وذلك بعدم تحديد كلمة مرور له) واستخدام sudo للحصول على امتيازات الجذر. يملك أول مستخدم في النظام امتيازات الجذر الكاملة باستخدامه للأمر sudo، الذي يستطيع أن يعطي باقي المستخدمين العاديين بعض الامتيازات.

تغيير المستخدم والمجموعة المالكة

يُستخدم الأمر chown لتغير المالك والمجموعة المالكة لملف أو مجلد. يجب استخدام امتيازات الجذر عند تنفيذ هذا الأمر. الشكل العام للأمر chown هو:

```
chown [owner][:[group]] file...
```

يُغيّر الأمر chown مالك أو المجموعة المالكة للملف بالاستناد إلى قيمة الوسيط الأول من الأمر. يحتوي الجدول الآتي على بعض الأمثلة:

الجدول 9-7: أمثلة عن وسائط الأمر chown

الوسيط	النتيجة
bob	تغيير ملكية الملف من المستخدم الحالي إلى المستخدم bob.
bob:users	تغيير ملكية الملف من المستخدم الحالي إلى المستخدم bob وتغيير المجموعة المالكة من المجموعة الحالية إلى المجموعة "users".
:admin	تغيير المجموعة المالكة إلى المجموعة "admin". لن يُغيّرَ مالك الملف.

bob:	تغيير مالك الملف من المستخدم الحالي إلى المستخدم bob، وتغيير المجموعة المالكة إلى المجموعة الافتراضية التي ينتمي إليها المستخدم bob (تكون عادةً بنفس اسم المستخدم، كما ذكرنا سابقًا)

لنفترض أن لدينا مُستخدمَين هما: "janet" الذي يملك امتيازات الجذر و "tony" الذي لا يملكها. وأراد المستخدم janet نسخ ملف من مجلد المنزل الخاص به إلى مجلد المنزل الخاص بالمستخدم tony. ولأن janet يريد من tony أن يستطيع تعديل الملف، فسيُغيّر janet ملكية الملف إلى المستخدم tony:

```
[janet@linuxbox ~]$ sudo cp myfile.txt ~tony
Password:
[janet@linuxbox ~]$ sudo ls -l ~tony/myfile.txt
-rw-r--r-- 1 root  root 8031 2008-03-20 14:30 /home/tony/myfile.txt
[janet@linuxbox ~]$ sudo chown tony: ~tony/myfile.txt
[janet@linuxbox ~]$ sudo ls -l ~tony/myfile.txt
-rw-r--r-- 1 tony  tony 8031 2008-03-20 14:30 /home/tony/myfile.txt
```

يمكننا ملاحظة أن المستخدم janet نَسَخَ الملف من مجلد المنزل الخاص به إلى مجلد المنزل الخاص بالمستخدم tony. ومن ثم نَقَلَ janet ملكية الملف من المستخدم الجذر (نتيجة استخدام الأمر sudo) إلى tony. وباستخدام النقطتَين الرأسيتَين، غيّرَ janet المجموعة المالكة إلى المجموعة tony أيضًا.

لاحظ أنه وبعد الاستخدام الأول للأمر sudo، فإن janet لم يُسأل عن كلمة مروره. وهذا لأن الأمر sudo، في أغلب التوزيعات، "يثق" بك لعدة دقائق.

تغيير المجموعة المالكة باستخدام chgrp

يُستخدم الأمر chown في أنظمة يونكس القديمة لتغيير المستخدم المالك للملف وليس المجموعة المالكة. لهذا السبب، كان الأمر chgrp يُستخدم لتغيير المجموعة المالكة. وهو يعمل كالأمر chown لكنه محدود أكثر.

التمرن على الأذونات

لقد تعلمنا آلية عمل الأذونات، فحان الوقت الآن لتجربتها! سنُوجد حلًّا لمشكلة شائعة ألا وهي إنشاء مجلد مشترك. لنفترض أن لدينا مُستخدمَين هما "bill" و "karen" ويريد كلاهما أن يشارك ألبومات الموسيقى الخاصة به في مجلد مشترك. سيخزن كلاهما ملفات الموسيقى بصيغتي Ogg أو MP3. يملك المستخدم bill امتيازات الجذر بالأمر sudo.

أول شيء علينا فعله هو إنشاء مجموعة تحتوي على المستخدمَين bill و karen كأعضاء فيها. سيُنشئ

bill المجموعة باستخدام مدير المستخدمين الرسومي، ويُضيف المستخدمَين bill و karen إليها:



الشكل 3: إنشاء مجموعة جديدة في واجهة غنوم

ومن ثم يُنشئ bill المجلد الذي سيحتوي على ملفات الموسيقى:

```
[bill@linuxbox ~]$ sudo mkdir /usr/local/share/Music
Password:
```

ولأن المستخدم bill يُعالج الملفات خارج مجلد المنزل الخاص به، فإنه سيحتاج إلى امتيازات الجذر. تكون ملكية المجلد وأذوناته بعد إنشائه كالآتي:

```
[bill@linuxbox ~]$ ls -ld /usr/local/share/Music
drwxr-xr-x 2 root root 4096 2008-03-21 18:05 /usr/local/share/Music
```

كما لاحظتَ، إن أذونات المجلد هي 755 ومملوك من قِبل root. لمشاركة هذا المجلد، سيحتاج bill إلى تغيير المجموعة المالكة وتغيير أذونات الوصول للمجموعة المالكة لتمكين الكتابة في المجلد:

```
[bill@linuxbox ~]$ sudo chown :music /usr/local/share/Music
[bill@linuxbox ~]$ sudo chmod 775 /usr/local/share/Music
[bill@linuxbox ~]$ ls -ld /usr/local/share/Music
drwxrwxr-x 2 root music 4096 2008-03-21 18:05 /usr/local/share/Music
```

ماذا حصل إلى الآن؟ قمنا إلى الآن بإنشاء المجلد /usr/local/share/Music الذي يملكه الجذر (root) وأعطينا أذونات القراءة والكتابة إلى المجموعة music التي تحتوي على المستخدمَين bill و karen. إذًا، يستطيع المُستخدمان bill و karen إنشاء الملفات في المجلد/usr/local/share/Music. يستطيع باقي المستخدمين عرض محتويات المجلد لكن ليس إنشاء الملفات فيه.

المشكلة الآن تكمن في أن الملفات والمجلدات التي تُنشأ في مجلد Music تكون بالامتيازات العادية للمستخدمَين bill و karen:

```
[bill@linuxbox ~]$ > /usr/local/share/Music/test_file
[bill@linuxbox ~]$ ls -l /usr/local/share/Music
-rw-r--r-- 1 bill  bill  0 2008-03-24 20:03 test_file
```

في الواقع هنالك مشكلتان لا واحدة. أولًا، قيمة umask في هذا النظام هي 0022 التي تمنع افتراضيًّا أعضاء المجموعة من الكتابة إلى الملفات المملوكة من قِبل مستخدم آخر. لن يُشكّل ذلك مشكلةً إذا كان المجلد يحتوي على ملفات فقط ولا يحتوي أيّة مجلدات فرعية. ولأن المجلد سيحتوي على ملفات الموسيقى التي غالبًا ما تُرتَّب حسب الفنانين والألبومات. فسيحتاج أعضاء المجموعة إلى إذن إنشاء الملفات داخل المجلدات الفرعية التي قام مستخدمون آخرون بإنشائها. لذا، سنحتاج إلى تغيير قيمة umask للمستخدمَين bill و karen إلى 0002.

المشكلة الثانية أن المجموعة المالكة لأي ملف مُنشأ من قِبل أحد الأعضاء هي المجموعة الافتراضية لذاك العضو وليست مجموعة music. يمكن حلّ تلك المشكلة بتعيين الإذن setgid للمجلد:

```
[bill@linuxbox ~]$ sudo chmod g+s /usr/local/share/Music
[bill@linuxbox ~]$ ls -ld /usr/local/share/Music
drwxrwsr-x 2 root music 4096 2008-03-24 20:03 /usr/local/share/Music
```

لنجرب الآن ولنرَ فيما إن كانت الأذونات الجديدة ستحل المشكلة أم لا. سيعين المستخدم bill قيمة umask إلى 0002 ويحذف الملف السابق ويُنشئ ملفًا ومجلدًا في المجلد Music:

```
[bill@linuxbox ~]$ umask 0002
[bill@linuxbox ~]$ rm /usr/local/share/Music/test_file
[bill@linuxbox ~]$ /usr/local/share/Music/test_file
[bill@linuxbox ~]$ /usr/local/share/Music/test_dir
[bill@linuxbox ~]$ ls -l /usr/local/share/Music
drwxrwsr-x 2 bill 4096 2008-03-24 20:24 test_dir
-rw-rw-r-- 1 bill    0 2008-03-24 20:22 test_file
```

```
[bill@linuxbox ~]$
```

أذونات الملفات والمجلدات صحيحة وتعطي الحق لأعضاء مجموعة music بإنشاء الملفات والمجلدات داخل مجلد Music.

تبقى مشكلة واحدة هي umask. فسيبقى الضبط اللازم لهذا الأمر حتى نهاية الجلسة فقط. سنتعلم طريقة جعل التغيرات على الأمر umask دائمة في الفصل 11.

تغيير كلمة المرور

آخر المواضيع التي سنشرحها في هذا الفصل هو طريقة ضبط كلمات المرور لك (ولباقي المستخدمين في حال كانت لك امتيازات الجذر). نستخدم الأمر passwd لتعيين أو تغيير كلمة المرور؛ الشكل العام له هو:

```
passwd [user]
```

اطبع الأمر passwd لتغيير كلمة المرور الخاصة بك. ستُسأل عن كلمة المرور القديمة والجديدة:

```
[me@linuxbox ~]$ passwd
(current) UNIX password:
New UNIX password:
```

يحاول الأمر passwd إلزامك على اختيار كلمات مرور قوية. هذا يعني أنه يرفض كلمات المرور القصيرة جدًا أو الشبيهة بكلمات المرور السابقة أو تحتوي على كلمات "من القاموس" أو سهلة التخمين:

```
[me@linuxbox ~]$ passwd
(current) UNIX password:
New UNIX password:
BAD PASSWORD: is too similar to the old one
New UNIX password:
BAD PASSWORD: it is WAY too short
New UNIX password:
BAD PASSWORD: it is based on a dictionary word
```

إذا كانت لديك امتيازات الجذر، فتستطيع تحديد اسم أحد المستخدمين كوسيط للأمر passwd كي تعيين كلمة المرور لذاك المستخدم. يوجد العديد من الخيارات متاحة للمستخدم الجذر للسماح بإقفال حساب أحد المستخدمين مؤقتًا أو تحديد فترة صلاحية كلمة المرور وغيرها. راجع صفحة الدليل للأمر passwd لمزيد من المعلومات.

الخلاصة

لقد تعلمنا في هذا الفصل آلية عمل الأذونات التي تُعطيها الأنظمة الشبيهة بِيونكس (كَلينُكس) للمستخدمين، من قراءة، وكتابة، وتنفيذ. تعود فكرة الأذونات إلى أولى أيام نظام يونكس وبقيت إلى يومنا هذا. لكن الأذونات في الأنظمة الشبيهة بِيونكس تفتقر إلى السهولة والتبسيط التي تتمتع بها الأذونات في الأنظمة الأحدث منها.

الفصل العاشر:

العمليات

تكون الأنظمة الحديثة عادةً متعددة المهام. أي أنها توهمك بأنها تقوم بأكثر من مهمة في آن واحد وذلك بالتبديل السريع ما بين البرامج التي تُنَفَّذ. يُنظِّم نظام لينُكس البرامج باستخدام "العمليات" (processes) لكي تنتظر دورها للمعالجة في وحدة المعالجة المركزية.

يُصاب الحاسوب في بعض الأحيان بالبطء أو يتوقف تطبيق ما عن الاستجابة. سنلقي في هذا الفصل نظرة على بعض الأدوات الموجودة في سطر الأوامر التي تسمح لنا بالاطلاع على الأشياء التي تقوم بها البرامج، وطريقة إنهاء العمليات التي لا تُنفِّذ عملها كما يجب.

سنشرح الأوامر الآتية في هذا الفصل:

- `ps` – إظهار العمليات التي تعمل حاليًا في النظام.
- `top` – إظهار المهام.
- `jobs` – إظهار قائمة بالمهام المُفعَّلة.
- `bg` – نقل مهمة إلى الخلفية.
- `fg` – إعادة المهمة من الخلفية.
- `kill` – إرسال إشارة إلى عملية.
- `killall` – قتل العمليات بتحديد اسمها.
- `shutdown` – إيقاف أو إعادة تشغيل النظام.

كيف تجري العمليات

تُهيّئ النواة عددًا من الأنشطة الخاصة بها على شكل "عمليات" (processes) عندما يبدأ النظام. ثم تبدأ برنامجًا يُدعى `init`. بدوره، يُشغِّل `init` سلسلةً من سكربتات الشِل (الموجودة في مجلد `/etc`) تسمى `init scripts`. التي تبدأ جميع خدمات النظام. العديد من تلك الخدمات تدعى daemon programs. أي البرامج التي تبقى في الخلفية وتنفذ مهامها دون أي تدخل من المستخدم. وحتى إن لم نسجل دخولنا، فسوف يكون النظام مشغولًا (ولو قليلًا) بالقيام ببعض الأعمال الروتينية.

إمكانية تشغيل برنامج لبرامج أُخرى يعبر عنه في تعابير العمليات بالعملية الأب (parent process) تُشكِّل

عمليات أبناء (child processes).

تدير النواة معلومات حول كل عملية للمساعدة في إبقائها منظمةً. على سبيل المثال، لكل عملية رقم خاص بها يسمى "رقم العملية" (process ID أو PID). تعطى تلك الأرقام للعمليات بترتيب تصاعدي، حيث يكون للعملية init الرقم 1 دائمًا. تتتبّع النواة أيضًا مقدار الذاكرة التي تعطى لكل عملية بالإضافة إلى قابلية العمليات على إكمال التنفيذ. وكما في الملفات، فإنه يوجد مستخدمين مالكين للعمليات ...إلخ.

مشاهدة العمليات

أكثر الأوامر استعمالًا لمشاهدة العمليات (توجد عدّة أوامر تقوم بذلك) هو ps. لبرنامج ps العديد من الخيارات، لكن في أبسط الصيغ يكون كالآتي:

```
[me@linuxbox ~]$ ps
  PID TTY           TIME CMD
 5198 pts/1     00:00:00 bash
10129 pts/1     00:00:00 ps
```

أظهرت نتيجة المثال السابق عمليتَين، العملية ذات الرقم 5198، والعملية ذات الرقم 10129، اللتين تُمثلان bash و ps على الترتيب. وكما لاحظنا، لا يعرض الأمر ps الكثير من المعلومات افتراضيًّا، فهو يعرض العمليات المرتبطة بجلسة الطرفية الحالية فقط. سنحتاج إلى إضافة بعض الخيارات لمشاهدة المزيد؛ لكن قبل القيام بذلك، لنلقِ نظرة على الحقول الأُخرى التي أُظهرت بالأمر ps. TTY هو اختصار للكلمة "Teletype" الذي يشير إلى الطرفية المتحكمة (controlling terminal) في العملية. يشير الحقل TIME إلى مقدار الوقت المستهلك من المعالج من قِبل العملية.

سنحصل على تصور أكبر عن ما يفعله النظام إذا أضفنا الخيار "x":

```
[me@linuxbox ~]$ ps x
   PID TTY   STAT     TIME COMMAND
  2799 ?     Ssl      0:00 /usr/libexec/bonobo-activation-server –ac
  2820 ?     Sl       0:01 /usr/libexec/evolution-data-server-1.10 --
 15647 ?     Ss       0:00 /bin/sh /usr/bin/startkde
 15751 ?     Ss       0:00 /usr/bin/ssh-agent /usr/bin/dbus-launch --
 15754 ?     S        0:00 /usr/bin/dbus-launch --exit-with-session
 15755 ?     Ss       0:01 /bin/dbus-daemon --fork --print-pid 4 –pr
 15774 ?     Ss       0:02 /usr/bin/gpg-agent -s –daemon
 15793 ?     S        0:00 start_kdeinit --new-startup +kcminit_start
```

```
15794 ?     Ss       0:00 kdeinit Running...
15797 ?     S        0:00 dcopserver —nosid


and many more...
```

إضافة الخيار "x" (لاحظ عدم وجود الشرطة التي تسبق عادةً الخيارات) يُخبر الأمر ps أن يعرض جميع العمليات دون الأخذ بعين الاعتبار أيّة طرفية (إذا كان هنالك واحدة) تُشَغِّلهم. يُشير الرمز "?" في حقل TTY إلى عدم وجود طرفية متحكمة في العملية. لقد استطعنا معرفة كل العمليات التي "نملكها" عندما استخدمنا هذا الخيار.

يعرض الأمر ps قائمةً طويلةً جدًا؛ لأن النظام يُشغّل عددًا كبيرًا من العمليات. لذا، فمن المفيد تمرير مخرجات الأمر ps إلى `less` لتسهيل مشاهدة النتائج. تنتج بعض الخيارات أيضًا أسطرًا طويلة؛ لذلك قد يكون من الجيد تكبير نافذة محاكي الطرفية.

أُضيفَ حقل جديد مُعَنون بالكلمة STAT إلى المخرجات. STAT هو اختصار للكلمة "state" أي حالة، ويكشف عن حالة العمليات الجارية حاليًا:

الجدول 10-1: حالات العمليات

الحالة	الشرح
R	قيد التنفيذ (Running). هذا يعني أن العملية قيد التنفيذ أو جاهزة لذلك.
S	النوم (sleeping). لا تُنَفّذ العملية حاليًا، بل هي متوقفة في انتظار حدوث شيءٍ ما كالضغط على أحد الأزرار أو وصول حزمة من الشبكة...
D	نوم غير قابل للمقاطعة (Uninterpretable Sleep). تنتظر العملية المخرجات أو المدخلات كالكتابة أو القراءة من القرص.
T	عملية متوقفة. العملية التي أُمرت بالتوقف. سنناقشها لاحقًا.
Z	عمليات منتهية (zombie). وهي العمليات الابن التي قد انتهت ولم يتم "تنظيفها" من قِبل العملية الأب.
<	عملية بأولوية قصوى. من الممكن إعطاء المزيد من الأهمية لإحدى العمليات، وذلك بمنحها المزيد من وقت المعالجة. تسمى أولوية العملية بالمصطلح "niceness" أو "اللطافة". تسمى العملية ذات الأولوية القصوى بأنها أقل لطافةً لأنها تستهلك المزيد من وقت المعالج، مما يؤدي إلى ترك وقت

	معالجة أقل لباقي العمليات.
N	عملية ذات أولوية متدنية. ستُرسَل العملية ذات الأولوية المتدنية (أي عملية "لطيفة" [nice]) إلى المعالج عند الانتهاء من معالجة العمليات ذات الأولوية المرتفعة.

يمكن أن تُتبَع حالة العملية بمحارف أخرى التي تشير إلى بعض الخصائص الغريبة للعمليات. راجع صفحة الدليل للأمر ps لمزيد من المعلومات.

مجموعة خيارات أُخرى مشهورة هي "aux" (دون أن تُسبق بشرطة) التي يمكن أن تعطينا المزيد من المعلومات:

```
[me@linuxbox ~]$ ps aux
USER     PID %CPU %MEM  VSZ     RSS TTY       STAT   START   TIME
COMMAND
root       1 0.0 0.0  2136     644 ?  Ss      Mar05   0:31   init
root       2 0.0 0.0     0       0 ?  S<      Mar05   0:00   [kt]
root       3 0.0 0.0     0       0 ?  S<      Mar05   0:00   [mi]
root       4 0.0 0.0     0       0 ?  S<      Mar05   0:00   [ks]
root       5 0.0 0.0     0       0 ?  S<      Mar05   0:06   [wa]
root       6 0.0 0.0     0       0 ?  S<      Mar05   0:36   [ev]
root       7 0.0 0.0     0       0 ?  S<      Mar05   0:00   [kh]


and many more...
```

تُظهر مجموعة الخيارات السابقة العمليات التي يشغلها أي مستخدم. يؤدي استخدام الخيارات دون طباعة شرطة قبلهم إلى تنفيذ الأمر "بسلوك" BSD. نسخة لينُكس من البرنامج ps تحاكي سلوك برنامج ps الموجود في مختلف الأنظمة الشبيهة بِيونكس. سنحصل على الخانات الآتية عند استخدام هذه الخيارات:

الجدول 10-2: ترويسات الحقول المُستخدمة في برنامج ps نمط BSD

الحقل	الشرح
USER	رقم مُعرّف (ID) المستخدم المالك للعملية.
%CPU	النسبة المئوية لاستخدام المعالج.
%MEM	النسبة المئوية لاستخدام الذاكرة.

VSZ	حجم الذاكرة الظاهرية (Virtual memory).
RSS	المقدار المستخدم من ذاكرة الوصول العشوائي الفيزيائية (RAM) التي تستخدمها العملية مُقدرًا بالكيلوبايت. اختصار للعبارة "Resident Set Size".
START	الوقت الذي بدأت فيه العملية، وسيُذكر التاريخ مكان القيم التي تتجاوز أربعًا وعشرين ساعة.

الاطلاع على العمليات تفاعليًّا مع الأمر top

على الرغم من أن الأمر ps يعرض معلومات كثيرة حول ما يجري في الحاسوب، إلا أنه يوفر "لقطة" عن حالة الجهاز في لحظة زمنية محددة عندما نُفِّذ الأمر ps. نستخدم الأمر top للحصول على نشاط الجهاز بعرض أكثر تفاعليةً:

```
[me@linuxbox ~]$ top
```

يقوم برنامج top بتحديث متواصل (كل ثلاث ثواني افتراضيًا) لقائمة بالعمليات التي تجري في النظام مرتبةً حسب نشاطها. أُطلقت الكلمة top على البرنامج لأنه يُستخدم لمعرفة أكثر العمليات استهلاكًا لمورد النظام. يتكون العرض الذي يوفره top من قسمَين: الأول هو ملخص عن حالة النظام في أعلى شاشة العرض، يتبعه القسم الثاني الذي هو جدول يحتوي على العمليات مرتبةً وفق استهلاكها للمعالج:

```
top - 14:59:20 up 6:30, 2 users, load average: 0.07, 0.02, 0.00
Tasks: 109 total, 1 running, 106 sleeping, 0 stopped, 2 zombie
Cpu(s): 0.7%us, 1.0%sy, 0.0%ni, 98.3%id, 0.0%wa, 0.0%hi, 0.0%si
Mem: 319496k total,  314860k used,     4636k free,  19392k buff
Swap:875500k total,  149128k used,   726372k free, 114676k cach
 PID    USER   PR  NI   VIRT   RES    SHR S %CPU %MEM    TIME+   COMMAND
 6244    me     39  19  31752  3124  2188 S   6.3  1.0 16:24.42  trackerd
11071    me     20   0   2304  1092   840 R   1.3  0.3  0:00.14  top
 6180    me     20   0   2700  1100   772 S   0.7  0.3  0:03.66  dbus-dae
 6321    me     20   0  20944  7248  6560 S   0.7  2.3  2:51.38  multiloa
 4955    root   20   0   104m  9668  5776 S   0.3  3.0  2:19.39  Xorg
    1    root   20   0   2976   528   476 S   0.0  0.2  0:03.14  init
    2    root   15  -5      0     0     0 S   0.0  0.0  0:00.00  kthreadd
    3    root   RT  -5      0     0     0 S   0.0  0.0  0:00.00  migratio
```

```
    4   root  15  -5      0     0     0 S  0.0  0.0  0:00.72  ksoftirq
    5   root  RT  -5      0     0     0 S  0.0  0.0  0:00.04  watchdog
    6   root  15  -5      0     0     0 S  0.0  0.0  0:00.42  events/0
    7   root  15  -5      0     0     0 S  0.0  0.0  0:00.06  khelper
   41   root  15  -5      0     0     0 S  0.0  0.0  0:01.08  kblockd/
   67   root  15  -5      0     0     0 S  0.0  0.0  0:00.00  kseriod
  114   root  20   0      0     0     0 S  0.0  0.0  0:01.62  pdflush
  116   root  15  -5      0     0     0 S  0.0  0.0  0:02.44  kswapd0
```

يحتوي الملخص على الكثير من المعلومات المفيدة؛ وهذا شرحها:

الجدول 10-3: الحقول المستخدمة في top

السطر	الحقل	الشرح
1	`top`	اسم البرنامج.
	`14:59:20`	الوقت الحالي.
	`up 6:30`	هذا يُسمى "uptime" أي الزمن المنصرم منذ آخر إقلاع للجهاز. في هذا المثال، يعمل النظام منذ ست ساعات ونصف الساعة.
	`2 users`	هنالك مستخدمان قاما بتسجيل دخولهما إلى النظام.
	`load average:`	يُشير "مقدار الحِمل" إلى عدد العمليات التي تنتظر بعض الوقت حتى تستطيع أن تُنفَذ. وهذا يعني أنه عدد العمليات التي في حالة تنفيذ وتستهلك المعالج. تُعرض ثلاث قيم لكي تظهر مقدار الحِمل في فترات زمنية مختلفة. أول قيمة هي المتوسط الحسابي للحِمل لآخر 60 ثانية، والتي تليها لآخر خمس دقائق والأخيرة لآخر ربع ساعة. القيم تحت 1.0 تعني أن الجهاز لم يكن مشغولًا في تلك الفترة.
2	`Tasks:`	ملخص عن عدد العمليات وحالاتهم المختلفة.
3	`Cpu(s):`	يشير هذا السطر إلى مقدار استهلاك العمليات للمعالج.
	`0.7%us`	يُستخدم 0.7% من المعالج لعمليات المستخدم (user processes). التي هي العمليات خارج النواة.

	1.0%sy	1.0% من المعالج يُستخدم لعمليات النظام (النواة).
	0.0%ni	0.0% من المعالج يُستخدم للعمليات "اللطيفة" (أي ذات أولوية متدنية).
	98.3%id	أي ما نسبته 98.3% من المعالج لا يُستخدم الآن.
	0.0%wa	0.0% من المعالج ينتظر الدخل/الخرج.
4	Mem:	إظهار حجم الذاكرة الفيزيائية RAM المُستخدمة.
5	Swap:	إظهار حجم ذاكرة التبديل swap (الذاكرة الظاهرية) المُستخدمة.

يقبل الأمر top عددًا من أوامر لوحة المفاتيح. أكثر أمرين مهمين هما الأمر h الذي يعرض شاشة المساعدة، والأمر q الذي ينهي برنامج top.

توفر بيئتا سطح المكتب الرئيسيتين برمجيات رسومية تُظهر معلومات بشكل شبيه ببرنامج top (تعمل تلك البرامج بشكل مشابه لتطبيق "إدارة المهام" في نظام ويندوز) لكنني أجد أن top أفضل من البرامج الرسومية لأنه أسرع ولا يستهلك الكثير من موارد الجهاز.

على أيّة حال، لا يجب أن يكون برنامج مراقبة أداء النظام مصدرًا لاستهلاك موارد الجهاز التي نسعى لمراقبتها!

التحكم في العمليات

بعد أن تعلمنا طريقة مراقبة العمليات، حان الوقت لكي نتحكم فيهم. سنستخدم برنامجًا صغيرًا للقيام بتجاربنا عليه هو xlogo. برنامج xlogo هو برنامج بسيط يأتي مع خادم العرض X (الخدمة التي تُظهر البرامج الرسومية على جهازنا) الذي يُظهِر نافذة بسيطة قابلة للتكبير والتصغير تحتوي على شعار X. لنجربه:

```
[me@linuxbox ~]$ xlogo
```

بعد إدخال الأمر، ستظهر نافذة تحتوي على الشعار في مكانٍ ما من الشاشة. قد يعرض البرنامج xlogo رسالة تحذير في بعض التوزيعات وتستطيع ببساطة أن تتجاهلها.

تلميحة: إذا لم يتوفر xlogo في نظامك، فجرّب استخدام gedit أو kwrite بدلًا منه.

يمكننا التأكد من أن xlogo يعمل جيدًا بإعادة تحجيم (تغيير أبعاد) النافذة. إذا أُعيد رسم الشعار في الحجم

الجديد، فإن البرنامج يعمل دون مشاكل.

لاحظ عدم توفير مِحث الصدفة مرة أُخرى؟ السبب هو أن الصدفة تتنظر إنهاء البرنامج xlogo، كما باقي البرامج التي جربناها حتى الآن. وإذا أغلقت نافذة xlogo، فسيعود المِحث إلى ما كان عليه.

إنهاء العمليات

لنجرب ماذا سيحدث إذا شغلنا xlogo مرّة أُخرى (أدخل الأمر xlogo ومن ثم تأكد من أن البرنامج يعمل بشكل صحيح) وضغطنا على Ctrl-c في نافذة الطرفية:

```
[me@linuxbox ~]$ xlogo
[me@linuxbox ~]$
```

الاختصار Ctrl-c يقوم بمقاطعة (interrupts) برنامجٍ ما. هذا يعني أننا نطلب "بتهذيب" من البرنامج أن يغلق نفسه. ستُغلق نافذة xlogo بعد الضغط على Ctrl-c، وسيعود إلينا المِحث.

يمكن إنهاء معظم (لكن ليس جميع) برامج سطر الأوامر بهذه الطريقة.

نقل عملية إلى الخلفية

لنفترض أننا نريد عودة المِحث دون أن ننهي البرنامج xlogo. نستطيع فعل ذلك بوضع البرنامج في "الخلفية" (background). تخيل أن لدى الطرفية "أمامية" (foreground) التي تحتوي على الأشياء الظاهرة للعيان كمِحث الصدفة) وخلفية. نُتبع الأمر برمز "&" لتشغيل برنامج ما مباشرةً في الخلفية:

```
[me@linuxbox ~]$ xlogo &
[1] 28236
[me@linuxbox ~]$
```

ستظهر نافذة xlogo وسيعود مِحث الصدفة بعد تنفيذ الأمر. لكن بعض الأرقام قد طُبِعَت أيضًا. هذه الرسالة هي جزء من ميزة في الصدفة تسمى التحكم في المهمات (job control). تخبرنا الصدفة في هذه الرسالة أننا بدأنا المهمة ذات الرقم 1 ("[1]") ويكون رقم العملية المسند إليها هو 28236. إذا جربنا الأمر ps، فسنشاهد العملية الجديدة:

```
[me@linuxbox ~]$ ps
  PID TTY          TIME   CMD
10603 pts/1     00:00:00   bash
28236 pts/1     00:00:00   xlogo
```

```
28239 pts/1     00:00:00   ps
```

تسمح آلية التحكم في المهمات الموجودة في الصدفة بمشاهدة جميع المهمات التي بدأناها من جلسة الطرفية. نستطيع مشاهدة القائمة باستخدام الأمر jobs:

```
[me@linuxbox ~]$ jobs
[1]+ Running    xlogo &
```

تشير النتيجة إلى أنه لدينا مهمة واحدة تعمل وهي المهمة ذات الرقم "1"، وكان الأمر المنفذ هو xlogo &.

استعادة العمليات من الخلفية

عندما تكون العملية في الخلفية فإنها لا تتفاعل أبدًا مع لوحة المفاتيح وحتى لا يمكن إنهاؤها بالضغط على Ctrl-c. لإعادة العمليات إلى الأمامية، نستخدم الأمر fg على هذا النحو:

```
[me@linuxbox ~]$ jobs
[1]+ Running    xlogo &
[me@linuxbox ~]$ fg %1
xlogo
```

استخدمنا الأمر fg يتبعه علامة النسبة المئوية ومن ثم رقم المهمة (عادةً ما يُسمى رقم المهمة بالاسم jobspec). إذا كانت لدينا مهمة واحدة في الخلفية؛ فبإمكاننا تنفيذ الأمر fg دون وسائط. اضغط على Ctrl-c لإنهاء xlogo.

إيقاف العمليات

قد تريد في بعض الأحيان إيقاف عملية دون أن تنهيها. عادة ما يتم إيقاف العمليات للسماح لها بالانتقال إلى الخلفية. لإيقاف عملية تعمل في "الأمامية" اضغط على Ctrl-z. لنجربها بطابعة xlogo ومن ثم Ctrl-z:

```
[me@linuxbox ~]$ xlogo
[1]+ Stopped    xlogo
[me@linuxbox ~]$
```

بإمكاننا التأكد من إيقاف xlogo بتغيير أبعاد نافذته. سنشاهد أنه متوقف تمامًا. يمكننا الآن استعادة البرنامج إلى الأمامية باستخدام الأمر fg أو نقله إلى الخلفية باستخدام الأمر bg:

```
[me@linuxbox ~]$ bg %1
[1]+ xlogo &
[me@linuxbox ~]$
```

وكما في أمر fg، فإن رقم المهمة اختياري إذا كانت هنالك مهمة واحدة فقط.

يمكن نقل العملية من الأمامية إلى الخلفية عند بدء برنامج رسومي من سطر الأوامر ونسيان نقله مباشرةً إلى الخلفية بالرمز &.

لكن لماذا نريد تشغيل برنامج رسومي من الطرفية؟ هنالك سببان لذلك: السبب الأول إذا أردنا تشغيل برنامج رسومي غير موجود في قوائم مدير النوافذ المستخدم (xlogo على سبيل المثال). السبب الثاني هو عند تشغيلك لبرنامج رسومي من الطرفية، فإنك تستطيع مشاهدة رسائل الخطأ غير الظاهرة إذا شُغِّل من الواجهة الرسومية مباشرةً. يفشل، في بعض الأحيان، تشغيل برنامج ما عندما نُشغِّله من الأيقونة الموجودة في القوائم، إلا أن تشغيله من سطر الأوامر يتيح لنا رؤية رسائل الخطأ ومعرفة مكمن المشكلة. بالإضافة إلى ذلك، لبعض البرامج الرسومية خيارات مفيدة ومثيرة للاهتمام نستطيع استخدامها عند تشغيل البرنامج من سطر الأوامر.

الإشارات

يُستخدم الأمر kill "لقتل" العمليات. مما يسمح لنا بإنهاء البرامج التي لا تستجيب. جرّب هذا المثال:

```
[me@linuxbox ~]$ xlogo &
[1] 28401
[me@linuxbox ~]$ kill 28401
[1]+ Terminated  xlogo
```

شغلنا، في البداية، xlogo في الخلفية. فطبعت الصدفة رقم المهمة و PID (رقم العملية). ومن ثم استخدمنا الأمر kill وحددنا رقم العملية المُراد إنهاؤها. يمكننا أيضًا تحديد العملية برقم مهمتها (على سبيل المثال "%1") وليس فقط رقم عمليتها (PID).

وعلى الرغم من أن استخدام الأمر kill سهل وبسيط؛ لكن يوجد الكثير من الأشياء التي يمكن فعلها أكثر من مجرد إنهاء العمليات. الأمر kill لا يقوم فعلًا "بقتل" البرامج بل يرسل إليهم إشارات (signals). الإشارات هي طريقة من الطرق التي يتعامل فيها النظام مع العمليات. لقد شاهدنا بالفعل استخدام الإشارات وذلك عند الضغط على Ctrl-c و Ctrl-z. عندما يتم الضغط على هذه الأزرار في الطرفية، فإنها تُرسل إشارة للبرنامج الذي يعمل في الأمامية. في حالة الاختصار Ctrl-c، فإنها ترسل إشارة تدعى INI (Interrupt)؛ أما مع Ctrl-z فهي ترسل إشارة تسمى TSTP (أي Terminal Stop). في المقابل، "تستمع" البرامج إلى تلك

الإشارات وتقوم بشيء ما عند تلقيها. في الواقع، إمكانية استماع البرامج للإشارات تسمح لها بأن تقوم بعدّة أشياء عند تلقيها إشارة الإنهاء كحفظ الملف الحالي ...إلخ.

إرسال الإشارات باستخدام kill

يُستخدم الأمر `kill` لإرسال الإشارات إلى البرامج. أكثر شكل عام شائع له هو:

```
kill [-signal] PID...
```

إذا لم تحدد الإشارة فستُعتبر TERM ‏(Terminate) افتراضيًّا. يُستخدم الأمر `kill` في أغلب الأحيان لإرسال الإشارات الآتية:

الجدول 10-4: الإشارات الشائعة

الرقم	الاسم	الشرح
1	HUP	إشارة التعليق (Hangup)، هذه الإشارة من بقايا الأيام الخالية التي كانت الطرفيات تُوصل فيها إلى الحواسيب البعيدة بواسطة المودمات وخطوط الهاتف. تُعلِم هذه الإشارة البرامج بأن الطرفية التي شغلتهم قد "علّقت". أثر هذه الإشارة هو إغلاق جلسة الطرفية الحالية وإنهاء البرامج التي تعمل في الأمامية. تُستخدم هذه الإشارة من العديد من "الخدمات" (daemons) لإعادة التهيئة. أي أن الخدمة التي تستقبل تلك الإشارة تُعيد تشغيل نفسها وتقرأ ملف الإعدادات من جديد. يقبل خادم أباتشي استخدام إشارة HUP.
2	INT	تقوم بنفس عمل الاختصار `Ctrl-c` الذي يُرسَل من الطرفية. ستنهي عادةً البرنامج.
9	KILL	القتل. هذه الإشارة من نوع خاص. بينما تُعالج البرامج الإشارات التي تُرسل إليها كلٌ بطريقته، بما فيها تجاهل الإشارة تمامًا، إلا أن الإشارة KILL لا تُرسَل إلى البرنامج المراد قتله أبدًا. بل ستُنهي النواة العملية مباشرةً. لكن عندما تُنهى العملية بهذه الطريقة، فإن العملية لا تملك الفرصة لكي تحفظ ملف العمل (على سبيل المثال). لهذا السبب لا يجب استخدام الإشارة KILL إلا كَحَلّ أخير عند فشل باقي الإشارات.

15	TERM	الإنهاء (Terminate). الإشارة الافتراضية التي يُرسلها الأمر kill.
18	CONT	الإكمال (Continue). أي استعادة العملية بعد إرسال إشارة STOP إليها.
19	STOP	الإيقاف (Stop). تؤدي هذه الإشارة إلى إيقاف العملية دون إنهائها. هذه الإشارة شبيهة بإشارة KILL حيث لا تُرسَل إلى البرنامج الهدف. لذا، ليس بإمكان البرنامج الهدف تجاهلها.

لنجرب الأمر kill:

```
[me@linuxbox ~]$ xlogo &
[1] 13546
[me@linuxbox ~]$ kill -1 13546
[1]+ Hangup  xlogo
```

شغلنا، في هذا المثال، البرنامج xlogo في الخلفية ومن ثم أرسلنا إليه الإشارة HUP بالأمر kill. سيتم إنهاء برنامج xlogo، وتخبرنا الصدفة أن العملية الموجودة في الخلفية قد استقبلت الإشارة HUP. ربما ستحتاج إلى الضغط على زر Enter عدة مرات حتى تظهر لك الرسالة. لاحظ أنه بإمكانك إرسال الإشارات إما بأرقامهم أو بأسمائهم، أو بأسمائهم مع إسباقها بالأحرف "SIG":

```
[me@linuxbox ~]$ xlogo &
[1] 13601
[me@linuxbox ~]$ kill -INT 13601
[1]+ Interrupt  xlogo
[me@linuxbox ~]$ xlogo &
[1] 13608
[me@linuxbox ~]$ kill -SIGINT 13608
[1]+ Interrupt  xlogo
```

أعد تنفيذ الأوامر السابقة مع إشارات أُخرى. تذكر أنه بإمكانك استخدام رقم المهمة بدلًا من رقم العملية.

وكما في الملفات، لدى العمليات مستخدمين مالكين، ويجب أن تكون مالك العملية (أو المستخدم الجذر) لكي تستطيع إرسال الإشارات باستخدام الأمر kill.

بالإضافة إلى الإشارات السابقة التي تُستخدم عادةً مع الأمر kill؛ يوجد هناك عدد من الإشارات التي يستخدمها النظام، يلخص الجدول الآتي أبرز تلك الإشارات:

الجدول 10-5: إشارات شائعة أُخرى

الرقم	الاسم	الشرح
3	QUIT	الخروج.
11	SEGV	انتهاك حصة الذاكرة (Segmentation Violation). تُرسَل هذه الإشارة إذا استخدم البرنامج الذاكرة استخدامًا غير شرعيًا، أي حاول الكتابة في مكان لا يُسمح له بالكتابة فيه.
20	TSTP	الإيقاف من قِبل الطرفية (Terminal Stop). تُرسَل هذه الإشارة من قِبل الطرفية عند الضغط على Ctrl-z وعلى النقيض من الإشارة STOP، بإمكان البرنامج تجاهل الإشارة TSTP.
28	WINCH	حدوث تغيير في نافذة البرنامج (Window Change). تُرسَل هذه الإشارة من قِبل النظام عند تغير أبعاد نافذة ما. تُعيد بعض البرامج، كبرنامجَيّ top و less، إظهار البيانات لكي تتسع في أبعاد النافذة الجديدة.

يمكن للمهتمين الحصول على قائمة كاملة بجميع الإشارات باستخدام الأمر:

```
[me@linuxbox ~]$ kill -l
```

إرسال الإشارات إلى أكثر من عملية بالأمر killall

يمكن أيضًا إرسال الإشارات إلى أكثر من عملية تُطابق برنامجًا معينًا أو اسم مستخدمٍ ما بالأمر killall. الشكل العام له:

```
killall [-u user] [-signal] name...
```

لمعرفة آلية عمل الأمر killall، سنُنشئ عدّة نوافذ من برنامج xlogo ومن ثم سنُنهيهم:

```
[me@linuxbox ~]$ xlogo &
[1] 18801
[me@linuxbox ~]$ xlogo &
[2] 18802
[me@linuxbox ~]$ killall xlogo
[1]- Terminated  xlogo
```

```
[2]+ Terminated  xlogo
```

تذكر أنك، وكما في الأمر kill، ستحتاج إلى امتيازات الجذر لإرسال الإشارات إلى عمليات لا تملكها.

المزيد من الأوامر المتعلقة بالعمليات

لما كانت مراقبة العمليات مهمةً أساسيةً لمدير النظام، فيتوفر عدد من الأوامر لذلك. يحتوي الجدول الآتي على أسماء بعضها:

الجدول 10-6: أوامر أُخرى متعلقة بالعمليات

الأمر	الشرح
pstree	عرض قائمة العمليات على نمط "شجرة" تُظهر علاقة الأب/الابن بين العمليات.
vmstat	إظهار لقطة لمقدار استهلاك النظام للموارد المختلفة بما فيها ذاكرة الوصول العشوائي وذاكرة التبديل ومعدل الإدخال/الإخراج للقرص. للحصول على عرض متواصل لهذه المعلومات، أتبعْ الأمر بالتأخير الزمني للتحديث مقدرًا بالثانية. على سبيل المثال: vmstat 5.
xload	برنامج رسومي يعرض الحِمل على النظام خلال فترة زمنية محددة.
tload	شبيه ببرنامج xload إلا أنه يُظهِر مخطط الحِمل في الطرفية.

الخلاصة

توفر أغلب الأنظمة الحديثة آليةً لإدارة العمليات المختلفة. يوفر نظام لينُكس مجموعةً من الأدوات لهذا الغرض. وذلك لأن لينُكس هو من أشهر الأنظمة التي تُشغّل الخوادم في العالم. لكن على النقيض من باقي الأنظمة، يعتمد لينُكس اعتمادًا كبيرًا على أدوات سطر الأوامر بدلًا من الأدوات الرسومية، يُفضَّل استخدام أدوات سطر الأوامر بسبب سرعتها وخفتها. وعلى الرغم من أن الأدوات الرسومية تكون ذات مظهر جميل، إلا أنها تُسبب حِملًا كثيرًا على النظام!

تُرِكَتْ هَذِهِ الصَفْحَةُ فَارِغَةً عَمْدًا

الباب الثاني:
الإعدادات والبيئة

الفصل الحادي عشر:

البيئة

كما ناقشنا سابقًا، تعالج الصدفة معلومات في جلستنا الحالية تسمى "البيئة" (Environment). تستخدم البرامج البيانات الموجودة في البيئة للحصول على بعض المعلومات عن الإعدادات التي نستخدمها. وعلى الرغم من أن أغلب البرامج تستخدم "ملفات الإعدادات" (configuration files) لتخزين إعداداتها، إلا أن بعض البرامج ستحتاج إلى معرفة بعض القيم الموجودة في البيئة لكي تستطيع تعديل سلوكها.

سنتعامل مع الأوامر الآتية في هذا الفصل:

- `printenv` – عرض قسم من، أو كل البيئة الخاصة بنا.
- `set` – ضبط خيارات الصدفة.
- `export` – تصدير البيئة إلى برامج محددة.
- `alias` – إنشاء أمر بديل.

ما الذي يُخزَّن في البيئة؟

تُخزِّن الصدفة نوعَين أساسيَين من المعلومات في البيئة، لكن مع استخدام الصدفة bash، أصبح من الصعب التمييز ما بين هذين النوعَين وهما: متغيرات البيئة ومتغيرات الصدفة (التي تضبطها bash). بالإضافة إلى المتغيرات، تخزن البيئة معلومات برمجية تسمى "الأوامر البديلة" و "دوال الشِل". شرحنا طريقة إنشاء الأوامر البديلة في الفصل الخامس. وسنشرح إنشاء دوال شِل (التي ترتبط مع برمجة سكربتات شِل) في الباب الرابع.

استكشاف البيئة

يمكننا مشاهدة محتويات البيئة إما باستخدام الأمر set المضمن في bash أو برنامج printenv. يسمح الأمر set بعرض متغيرات البيئة والصدفة، بينما يظهر printenv متغيرات البيئة فقط. يُنصح بتمرير المخرجات إلى الأمر less، لأن محتويات البيئة ستشكل قائمة طويلة:

```
[me@linuxbox ~]$ printenv | less
```

يؤدي تنفيذ الأمر السابق إلى إظهار مخرجات شبيهة بالمخرجات الآتية:

```
KDE_MULTIHEAD=false
SSH_AGENT_PID=6666
HOSTNAME=linuxbox
GPG_AGENT_INFO=/tmp/gpg-PdOt7g/S.gpg-agent:6689:1
SHELL=/bin/bash
TERM=xterm
XDG_MENU_PREFIX=kde-
HISTSIZE=1000
XDG_SESSION_COOKIE=6d7b05c65846c3eaf3101b0046bd2b00-
1208521990.996705-1177056199
GTK2_RC_FILES=/etc/gtk-2.0/gtkrc:/home/me/.gtkrc-
2.0:/home/me/.kde/share/config/gtkrc-2.0
GTK_RC_FILES=/etc/gtk/gtkrc:/home/me/.gtkrc:/home/me/.kde/share/confi
g/gtkrc
GS_LIB=/home/me/.fonts
WINDOWID=29360136
QTDIR=/usr/lib/qt-3.3
QTINC=/usr/lib/qt-3.3/include
KDE_FULL_SESSION=true
USER=me
LS_COLORS=no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01
:cd=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe
:
```

ما شاهدناه هو قائمة بمتغيرات البيئة مع قيمهم. على سبيل المثال، شاهدنا متغيرًا باسم USER الذي تكون قيمته هي "me". يسمح الأمر printenv أيضًا بعرض قيمة أحد المتغيرات:

```
[me@linuxbox ~]$ printenv USER
me
```

عندما يُستخدم الأمر set دون خيارات أو وسائط، فإنه يعرض قائمة مرتبة أبجديًا (على النقيض من الأمر printenv) بمتغيرات البيئة والصدفة بالإضافة إلى دوال الشِل:

```
[me@linuxbox ~]$ set | less
```

من الممكن أيضًا عرض محتويات متغيرٍ ما بالأمر echo كالآتي:

```
[me@linuxbox ~]$ echo $HOME
/home/me
```

العنصر الوحيد الذي لا يُظهره الأمران set أو printenv هو الأوامر البديلة (aliases). نستخدم الأمر alias دون وسائط لإظهارهم:

```
[me@linuxbox ~]$ alias
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot
--show-tilde'
```

بعض المتغيرات المهمة

تحتوي البيئة على عدد كبير من المتغيرات. وعلى الرغم من أن البيئة الخاصة بك قد تختلف عن تلك التي عُرِضَت سابقًا، إلا أنك ستشاهد المتغيرات الآتية في بيئتك:

الجدول 11-1: متغيرات البيئة

المتغير	الشرح
DISPLAY	اسم شاشة العرض إذا كنت تشغل واجهة رسومية. تكون قيمة هذا المتغير عادةً هي ":0" التي تعني شاشة العرض الأولى التي أُنشئت من قِبل الخادم X.
EDITOR	اسم البرنامج المُستخدَم لتعديل الملفات النصية.
SHELL	اسم الصدفة التي تستخدمها.
HOME	مسار مجلد المنزل الخاص بك.
LANG	تحديد اللغة وترميز المحارف الخاص بك.
OLD_PWD	مجلد العمل السابق.
PAGER	اسم البرنامج الذي يُستخدم لعرض المخرجات على شكل صفحات، تكون قيمته غالبًا هي /usr/bin/less.

PATH	قائمة بالمجلدات التي سيُبحَث فيها عن الملفات التنفيذيّة للأوامر التي ندخلها، مفصولةً بنقطتين رأسيتين.
PS1	عبارة المِحث الأولى (prompt string 1). يحتوي هذا المتغير على محتويات مِحث الصدفة الخاص بك. وكما سنرى في فصل قادم، المِحث قابل للتخصيص كثيرًا.
PWD	مجلد العمل الحالي.
TERM	نوع الطرفية التي تستخدمها. تدعم الأنظمة الشبيهة بِيونكس العديد من بروتوكولات الطرفيات؛ يُستخدم هذا المتغير لتحديد نوع البروتوكول المستخدم في محاكي الطرفية الخاص بك.
TZ	تحديد المنطقة الزمنية الخاصة بموقعك. أغلب الأنظمة الشبيهة بِيونكس تتعامل مع الوقت بالتوقيت العالمي UTC ومن ثم تُظهر الوقت المحلي بعد إضافة مقدار من الزمن يعتمد على قيمة هذا المتغير.
USER	اسم المستخدم الخاص بك

لا تقلق إن لم تجد جميع القيم السابقة لأنها تختلف من توزيعة لأخرى.

كيف تعمل البيئة؟

عندما نُسجّل دخولنا إلى النظام، فإن الصدفة bash تبدأ وتقرأ سلسلة من سكربتات الإعدادات التي تسمى "ملفات بدء التشغيل" (startup files). التي تحدد البيئة الافتراضية المشتركة بين جميع المستخدمين. ومن ثم تُتبَع بقراءة المزيد من ملفات بدء التشغيل الموجودة في مجلد المنزل الخاص بنا والتي تُحَدِد بدورها البيئة الخاصة بنا. يختلف ترتيب قراءة تلك الملفات بين نوعين من جلسات الصدفة هما: جلسة الصدفة التي تحتاج إلى تسجيل الدخول (login shell session) والجلسة التي لا تحتاج إلى تسجيل الدخول (non-login shell session).

الجلسة التي تحتاج إلى تسجيل الدخول هي الجلسة التي نُطالب فيها بتوفير اسم المستخدم وكلمة المرور؛ كالتي تظهر عند تشغيل طرفية وهمية. أما الجلسات التي لا نحتاج فيها إلى تسجيل دخول هي في الغالب الجلسات التي نبدأها عند تشغيل الطرفية في الواجهة الرسومية.

الجلسات التي تحتاج إلى تسجيل الدخول تقرأ واحدًا أو أكثر من ملفات بدء التشغيل الموجودة في الجدول الآتي:

الجدول 11-2: ملفات البدء للصدفات التي تحتاج إلى تسجيل دخول

الملف	المحتوى
/etc/profile	سكربت إعدادات عام يطبق على جميع المستخدمين.
~/.bash_profile	ملف البدء الخاص بمستخدمٍ ما. يمكن أن يقوم بتوسيع أو استبدال الإعدادات الموجودة في ملف الإعدادات العام.
~/.bash_login	ستحاول bash قراءة هذا الملف إن لم يُعثَر على ملف ~/.bash_profile .
~/.profile	سيُقرأ هذا الملف إذا لم يكن الملفان ~/.bash_profile أو ~/.bash_login موجودَين. وهو السلوك الافتراضي للتوزيعات المبنية على دبيان كأوبنتو.

تقرأ الجلسات التي لا تحتاج إلى تسجيل الدخول الملفات الآتية:

الجدول 11-3: ملفات البدء للصدفات التي لا تحتاج إلى تسجيل دخول

الملف	المحتوى
/etc/bash.bashrc	سكربت الإعدادات العام لجميع المستخدمين.
~/.bashrc	ملف البدء الخاص بمستخدمٍ ما. يمكن أن يقوم بتوسيع أو استبدال الإعدادات الموجودة في ملف الإعدادات العام.

إضافةً إلى الملفات السابقة، فإن الجلسة التي لا تحتاج إلى تسجيل الدخول "ترث" البيئة من العملية الأب لها (تكون غالبًا جلسة تحتاج إلى تسجيل الدخول).

ألقِ نظرة على نظامك كي تعرف أي من الملفات السابقة موجود لديك. تذكر أن أغلب أسماء الملفات السابقة تبدأ بنقطة أي أنها ملفات مخفية. لذا، ستحتاج إلى استخدام الخيار "a-" عندما تستخدم الأمر ls.

ربما يكون الملف ~/.bashrc هو أهم ملف بالنسبة إلى المستخدم العادي، لأنه سيُقرأ دائمًا. حيث تقرأه الجلسات التي لا تحتاج إلى تسجيل دخول افتراضيًا. وأغلب ملفات بدء التشغيل التي تُنفّذ عند إنشاء جلسة تحتاج إلى تسجيل دخول تتضمن آلية لتنفيذ الأوامر الموجودة في ملف ~/.bashrc أيضًا.

ما هي ملفات البدء؟

إذا ألقينا نظرةً على ملف ‎.bash_profile اعتيادي (مأخوذ من نظام CentOS 4) فإنه سيكون على الشكل الآتي:

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin
export PATH
```

الأسطر التي تبدأ بالرمز "#" هي تعليقات ولا تُقرأ من قبل الصدفة. تُستخدم التعليقات لزيادة وضوح قراءة النص من قِبل البشر. أول الأشياء المثيرة للاهتمام تبدأ في السطر الرابع أي في الكود الآتي:

```
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi
```

الأسطر السابقة تسمى الدالة الشرطية if. سنشرحها شرحًا موسَّعًا عندما نناقش كتابة سكربتات الشِل في الباب الرابع، لكن يمكن ترجمتها الآن إلى العبارات الآتية:

```
If the file "~/.bashrc" exists, then
  read the "~/.bashrc" file.
```

لقد عرفت الآن كيف تتمكن الجلسة التي تحتاج إلى تسجيل الدخول من قراءة الملف ".bashrc". الشيء الآخر الذي يفعله ملف البدء هو تحديد قيمة المتغير PATH.

هل فكرت من قبل بالآلية التي تعرف الصدفة فيها مكان الملفات التنفيذية للأوامر التي ندخلها؟ على سبيل المثال، عند إدخالنا للأمر ls، فلن تبحث الصدفة في حاسوبنا بأكمله للعثور على الملف /bin/ls (المسار الكامل للملف التنفيذي للأمر ls)، بل تبحث في مجموعة محددة من المجلدات تُعرّف بواسطة المتغير PATH.

تحدد قيمة المتغير PATH عادةً (لكن ليس دائمًا، وذلك بالاعتماد على التوزيعة التي تستخدمها) في الملف /etc/profile باستخدام هذا الكود:

```
PATH=$PATH:$HOME/bin
```

عُدّلَ المتغير PATH لكي يضيف المجلد $HOME/bin إلى نهاية القائمة. هذا مثال عن توسعة المعاملات التي ناقشناها في الفصل السابع. جرّب المثال الآتي لإزالة الغموض:

```
[me@linuxbox ~]$ foo="This is some "
[me@linuxbox ~]$ echo $foo
This is some
[me@linuxbox ~]$ foo=$foo"text."
[me@linuxbox ~]$ echo $foo
This is some text.
```

بإضافة السلسلة النصية $HOME/bin إلى نهاية قيمة المتغير PATH، فسيُضاف المجلد $HOME/bin إلى قائمة المجلدات التي سيُبحث فيها عند تنفيذ أحد الأوامر. هذا يعني أنه إذا أردنا إنشاء مجلد داخل مجلد المنزل يحتوي على البرامج الخاصة بنا، فإن الصدفة ستساعدنا في ذلك وكل ما علينا فعله هو تسمية ذاك المجلد باسم bin.

ملاحظة: تستخدم أغلب التوزيعات قيمة متغير PATH الموجودة في المثال السابق. لكن بعض التوزيعات المبنية على دبيان كأوبنتو تختبر وجود المجلد ~/bin أولًا ومن ثم تضيفه تلقائيًا إلى المتغير PATH إذا كان موجودًا.

وفي النهاية نجد:

```
export PATH
```

يخبر الأمر export الصدفة بأن تجعل محتويات المتغير PATH متاحةً إلى العمليات الأبناء لتلك الصدفة.

تعديل البيئة

بعد أن تعرفنا على مكان تخزين ملفات البدء وعلى ماذا تحتوي، أصبح بإمكاننا أن نعدل فيهم لتخصيص البيئة.

أيّة ملفات يجب تعديلها؟

كقاعدة عامة، إضافة المجلدات إلى متغير PATH أو تعريف متغيرات بيئة جديدة يتم في ملف .bash_profile (أو الملف المكافئ له وذلك حسب التوزيعة التي تستخدمها. على سبيل المثال، تستخدم توزيعة أوبنتو الملف .profile). لأي شيء آخر، ضع التعديلات في ملف ".bashrc". اجعل التعديلات

مقتصرةً على الملفات الموجودة في مجلد المنزل لديك إلا إذا كنت مديرًا للنظام. فمن المؤكد أنك تستطيع تعديل الملفات الموجودة في مجلد /etc كملف profile، وذلك مفيد في عددٍ كبيرٍ من الحالات، لكن حاليًا، جرّب فقط على الملفات الخاصة بك كي لا تلحق أيّة أضرار بالنظام.

المحررات النصية

لتحرير (أو تعديل) ملفات البدء، ومختلف ملفات الإعدادات الأخرى؛ نستخدم برنامجًا يسمى محرر النصوص (text editor). محرر النصوص هو برنامج شبيه بالمحررات المكتبية (word processors) في أنه يسمح بتعديل الكلمات في الملفات وتحريك المؤشر ...إلخ. إلا أنه يختلف عن المحررات المكتبية في حفظه للملف كنص فقط دون أي تنسيق. وغالبًا ما يحتوي على ميزات تساعد في كتابة البرامج. المحررات النصية هي أداة مهمة جدًا لمطوري البرمجيات الذين يستخدمونها لكتابة الأكواد، أو لمدراء الأنظمة لتعديل ملفات الإعدادات التي تتحكم في سلوك النظام.

يوجد العديد من المحررات النصية لنظام لينُكس؛ غالب الظن أن نظامك يحتوي على عدّة محررات مثبّتة عليه. لماذا يوجد العديد من المحررات؟ ربما لأن المبرمجين يحبون كتابتها، ولأن المبرمجين يستخدمونها كثيرًا، فإنهم يطورونها للتحكم في إمكانياتها وسلوكها كما يريدون.

تُقسّم المحررات النصية إلى قسمَين: المحررات النصية المستخدمة في الواجهة الرسومية وتلك المستخدمة في سطر الأوامر. تحتوي غنوم وكدي بعض المحررات الرسومية الشهيرة. تحتوي واجهة غنوم على محرر gedit الذي يُسمى في القوائم بالاسم "Text Editor". أما كدي فتأتي مع ثلاثة محررات التي هي (بالترتيب من ناحية التعقيد) kedit و kwrite و kate.

يوجد هنالك العديد من المحررات التي تعمل من سطر الأوامر (text-based editors). أشهرها هو nano و vi و emacs. محرر nano هو محرر بسيط سهل الاستخدام صُمِمَ ليكون بديلًا عن محرر pico المستخدم من قِبل حزمة البريد الإلكتروني PINE. المحرر vi (أُستبدل في أغلب توزيعات لينُكس ببرنامج يُدعى vim، وهو اختصار للعبارة "Vi IMproved") هو المحرر التقليدي في أغلب الأنظمة الشبيهة بِيونكس وهو موضوع الفصل القادم. كُتِبَ المحرر emacs من قِبل ريتشارد ستالمان وهو بيئة تطوير برامج ضخمة، لجميع أغراض التحرير، وتقوم بكل شيء. وعلى الرغم من أنه متاح للتثبيت، إلا أنه من النادر أن تُثبّته التوزيعات افتراضيًا.

استخدام محرر نصي

يمكن استخدام المحررات النصية من الطرفية بكتابة اسم البرنامج يتبعه اسم الملف الذي تريد تعديله. سيعتبر المحرر أنك تريد إنشاء ملف جديد إن لم يكن الملف موجود مسبقًا. هذا مثال عن استخدام المحرر gedit:

```
[me@linuxbox ~]$ gedit some_file
```

يُشغِّل الأمر السابق محرر gedit ويفتح الملف المُسمى "some_file" إذا كان موجودًا.

تشرح أغلب المحررات الرسومية نفسها بنفسها، لذا لن نشرح طريقة استخدامها هنا. وإنما سنصب جُلّ اهتمامنا على المحررات التي تعمل من سطر الأوامر. سنبدأ مع nano. لنعدّل الملف bashrc. في محرر nano. لكن قبل ذلك، لنمارس بعض عادات التعديل "الآمنة". إذا ما أردنا تعديل ملف إعدادات مهم، فيجب علينا أخذ نسخة احتياطية من الملف أولًا. هذا سيحمينا في حال أحدثنا بعض "الفوضى" في الملف عند تحريره. لإنشاء نسخة احتياطية من ملف ".bashrc":

```
[me@linuxbox ~]$ cp .bashrc .bashrc.bak
```

لا يهم ما الاسم الذي ستطلقه على ملف النسخة الاحتياطية، لكن اختر اسمًا مفهومًا. الامتدادات ".bak" و ".sav" و ".old" و ".orig" هي امتدادات شهيرة للإشارة إلى ملف احتياطي. تذكر أيضًا أن الأمر cp سيستبدل الملفات دون سابق إنذار!

نستطيع الآن البدء في التعديل بعد أن أخذنا نسخةً احتياطيةً:

```
[me@linuxbox ~]$ nano .bashrc
```

ستحصل على شاشة شبيهة بالشاشة الآتية عندما يبدأ محرر nano:

```
   GNU nano 2.0.3                   File: .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
          . /etc/bashrc
fi

# User specific aliases and functions
```

```
                           [ Read 8 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell
```

ملاحظة: إن لم يوفر نظامك مُحرر nano فبإمكانك استخدام أي محرر رسومي عوضًا عنه.

تحتوي الشاشة على ترويسة في الأعلى، والنص الذي يتم تحريره في المنتصف، وقائمة بالأوامر في الأسفل. ولما كان محرر nano قد صُمِّمَ لاستبدال المحرر النصي الذي يأتي مع عميل بريد إلكتروني، فبكل تأكيد ستكون ميزاته بسيطة وقليلة نسبيًا.

أول الأوامر التي يجب أن تتعلمها عند التعامل مع أي محرر نصوص هي طريقة الخروج منه. في حالة nano، تستطيع الخروج من المحرر بالضغط على `Ctrl-x`. كما تشير إلى ذلك القائمة في الأسفل. الرمز "^X" يعني `Ctrl-x` يُرمز عادةً إلى رمز التحكم `Ctrl` بذاك الرمز في العديد من البرامج.

الأمر الثاني الذي يجب أن نتعلمه هو كيفية حفظ ما عدّلناه، نقوم بذلك في nano بالضغط على `Ctrl-o`. نحن الآن جاهزون لبعض التعديلات. حرّك المؤشر إلى نهاية الملف باستخدام السهم السفلي أو زر PageDown. ومن ثم أضف الأسطر الآتية:

```
umask 0002
export HISTCONTROL=ignoredups
export HISTSIZE=1000
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
```

ملاحظة: يمكن أن تكون التوزيعة قد ضبطت بعض تلك الأوامر الأوامر مُسبقًا، لكن تكرارها لن يضرّ أبدًا.

يشرح الجدول الآتي معنى الإضافات السابقة:

الجدول 11-4: الإضافات إلى ملف bashrc.

السطر	النتيجة
`umask 0002`	تحديد قيمة umask لحلّ مشاكل الأذونات في المجلدات المشتركة، كما ناقشنا في الفصل التاسع.
`export HISTCONTROL=ignoredups`	يؤدي إلى جعل خاصية تسجيل تأريخ الأوامر تتجاهل

	الأمر إذا كان موجود مسبقًا (أي أن الأمر مكرر).
`export HISTSIZE=1000`	زيادة عدد الأسطر التي ستُخزّن في تأريخ الأوامر من 500 إلى 1000.
`alias l.='ls -d .* --color=auto'`	إنشاء أمر جديد يدعى "`l.`" يظهر جميع محتويات المجلد التي تبدأ بنقطة (أي أنها ملفات مخفية).
`alias ll='ls -l --color=auto'`	إنشاء أمر جديد يُدعى "`ll`" يعرض محتويات المجلد بصيغة العرض التفصيلية.

كما لاحظت، ليست جميع التعديلات التي قمنا بها سهلة الفهم وواضحة؛ لذا، يكون من المفيد إضافة بعض التعليقات إلى ملف `.bashrc` لشرح الغاية من تلك التعديلات. باستخدام المحرر النصي، عدّل الإضافات لتصبح كالآتي:

```
# Change umask to make directory sharing easier
umask 0002

# Ignore duplicates in command history and increase
# history size to 1000 lines
export HISTCONTROL=ignoredups
export HISTSIZE=1000

# Add some helpful aliases
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
```

أفضل بكثير! الآن بعد إكمال التعديلات على الملف، لنقم بحفظه بالضغط على `Ctrl-o` ومن ثم الخروج من nano بالضغط على `Ctrl-x`.

لماذا التعليقات مهمة؟

عندما تُعدّل ملفات الإعدادات؛ فمن الضروري إضافة بعض التعليقات لشرح التغييرات التي قُمتَ بها. ستتذكر بكل تأكيد التعديلات التي قُمتَ بها في صباح الغد، لكن ماذا عن ستة أشهر من الآن؟ قدّم لنفسك خدمة وأضف بعض التعليقات. إنشاء سجل بالتغييرات التي قمت بها ليس فكرة سيئة على

> الإطلاق!
>
> قد تستخدم ملفات البدء التي تستخدمها bash وسكربتات الشِل الرمز "#" للإشارة إلى بدء التعليق. قد تستخدم ملفات إعدادات أُخرى رمزًا آخر، لكن أغلب ملفات الإعدادات تقبل بوجود تعليقات فيها. استخدمها كدليل لك.
>
> عادةً ما تشاهد بعض الأسطر في ملفات الإعدادات قد أُدرجت في تعليق لإيقاف تأثيرهم دون الحاجة إلى حذفهم. تُستخدم هذه الطريقة عادةً لإعطاء قارئ الملف بعض الخيارات الممكنة أو كمثال عن الشكل الصحيح لمحتويات الملف. على سبيل المثال، يحتوي الملف .bashrc في توزيعة أوبنتو على الأسطر الآتية:
>
> ```
> # some more ls aliases
> #alias ll='ls -l'
> #alias la='ls -A'
> #alias l='ls -CF'
> ```
>
> إذا أزلت رمز "#" من بداية آخر ثلاثة أسطر (تدعى هذه العملية بإزالة التعليق)، فإنك ستُفعّل تلك الأوامر البديلة. وإذا أضفت رمز "#" في بداية سطر، فإنك ستُعطل سطر الإعدادات دون حذفه.

تفعيل التغيرات التي قمنا بها

لن تُنفّذ التغيرات التي قمنا بها في ملف .bashrc إلا إذا بدأنا جلسة طرفية جديدة، لأن الملف .bashrc لا يُقرأ إلا في بداية الجلسة. لكننا نستطيع إجبار bash على إعادة قراءة ملف .bashrc المعدل بتنفيذ الأمر الآتي:

```
[me@linuxbox ~]$ source .bashrc
```

بعد القيام بذلك، نستطيع مشاهدة تأثير التغييرات التي قمنا بها. على سبيل المثال، لنجرب أحد الأوامر البديلة التي أنشأناها:

```
[me@linuxbox ~]$ ll
```

الخلاصة

لقد تعلمت في هذا الفصل مهارةً مهمةً جدًا ألا وهي تعديل ملفات الإعدادات باستخدام محرر نصي. اقرأ الآن صفحات الدليل للأوامر، لاحظ متغيرات البيئة التي تدعمها. سنتعلم المزيد عن كتابة دوال شِل في فصولٍ

لاحقة، التي يمكن تضمينها أيضًا في ملفات البدء للصدفة bash لإنشاء أوامر خاصة.

الفصل الثاني عشر:

مقدمة عن محرر vi

ليس سطر الأوامر مهارة تتعلمها في الصباح! بل يحتاج إلى سنوات من التدريب والممارسة. سنتعرف في هذا الفصل على محرر vi (يلفظ "ڤي آي")، وهو أحد البرامج الأساسية في يونكس. vi مشهور بواجهة المستخدم الصعبة، لكن عندما تشاهد محترف يستخدم vi، تجده يطبع على لوحة المفاتيح ويبدأ "بالعزف". لن نصبح محترفين في هذا الفصل، لكن عندما ننتهي، سنكون قادرين على إجراء المهمات الأساسية فيه.

لماذا علينا تعلم vi

في العصر الحديث الذي يوجد فيه المحررات الرسومية والمحررات التي تعمل سطر الأوامر ذات الاستخدام السهل كمحرر nano، لماذا علينا تعلم vi؟ حسنًا، توجد ثلاثة أسباب مهمة لذلك:

- محرر vi متوفر دائمًا. سينقذ حياتك إذا كان لديك نظام بدون واجهة رسومية، عند دخولك مثلًا إلى خادم بعيد أو على نظام محلي لا يعمل عليه خادم X بشكل صحيح (نتيجة حدوث أخطاء في ملفات الإعدادات). بينما يزداد انتشار nano، إلا أنه ليس محررًا عالميًا. يتطلب POSIX (معيار لتوافقية البرامج على أنظمة يونكس) وجود محرر vi.
- محرر vi خفيف وسريع. من الأسهل تشغيل vi بالمقارنة مع البحث عن المحرر الرسومي في القوائم ومن ثم انتظار بضعة ميغابايتات من الذاكرة كي تجهز. بالإضافة لذلك، فإن vi مصمم لسرعة الطباعة. كما سنرى، لا يرفع مستخدم vi ماهر يدَيّه عن لوحة المفاتيح عند الطباعة.
- لا نريد أن يظن مستخدمو لينُكس ويونكس الآخرون أننا مبتدئون.

حسنًا، ربما سببان مهمان فقط.

القليل من التاريخ

كُتب أول إصدار من vi في 1976 من قِبل Bill Joy. الذي هو طالب في جامعة كالفورنيا الذي شارك لاحقًا في تأسيس شركة Sun Microsystems. أخذ vi اسمه من كلمة "visual" أي مرئي. لأنه سمح بالتعديل على الطرفيات باستخدام مؤشر متحرك. قبل "المحررات المرئية" كان هنالك "المحررات السطرية" (line editors) التي لا تعالج أكثر من سطر في آن واحد وتصف التغيرات التي ستُجرى كالإضافة والحذف. اندمج vi مع محرر سطري يدعى ex، وبالتالي نستطيع تنفيذ أوامر التعديلات السطرية أثناء استخدامنا

لمحرر vi.

لا تحتوي أغلب توزيعات لينُكس على محرر vi الأصلي؛ بل تحتوي على بديل مطور يسمى vim (اختصار للعبارة "Vi IMproved") الذي كُتِبَ من قِبل Bram Moolenaar. يحتوي vim على ميزات مهمة مُضافة إلى محرر vi التقليدي. وعادةً ما يُنشأ أمر بديل للمحرر vim باسم "vi" في أنظمة لينُكس. سنعتبر وجود برنامج اسمه vi على جهازك والذي هو في الحقيقة vim.

بدء وإيقاف vi

استخدم الأمر الآتي لتشغيل vi:

```
[me@linuxbox ~]$ vi

~                    VIM - Vi IMproved
~
~                       version 7.1.138
~                 by Bram Moolenaar et al.
~         Vim is open source and freely distributable
~
~                 Sponsor Vim development!
~        type  :help sponsor<Enter>    for information
~
~        type  :q<Enter>               to exit
~        type  :help<Enter>  or  <F1>  for on-line help
~        type  :help version7<Enter>   for version info
~
~              Running in Vi compatible mode
~        type  :set nocp<Enter>        for Vim defaults
~        type  :help cp-default<Enter> for info on this
~
~
~
~
```

وكما فعلنا سابقًا مع محرر nano، فإن أول ما سنتعلمه هو طريقة إنهاء المحرر. نُدخِل الأمر الآتي لإنهاء vi (لاحظ أن النقطتين الرأسيتين ":" هما جزء من الأمر):

```
:q
```

سيعود مِحث الصدفة مرّة أُخرى. إذا لم ينتهِ برنامج vi (وذلك بسبب إدخالنا لتعديلٍ ما على الملف ولم يُحفظ ذاك التعديل)، فبإمكاننا أن نخبر vi أننا نعي ما نفعل وذلك بإضافة علامة تعجب بعد الأمر:

```
:q!
```

تلميحة: إذا "ضِعتَ" في vi، فجرب الضغط على زر Esc مرتَين حتى تعرف طريقك.

وضع التوافقية

شاهدنا الرسالة "Running in Vi compatible mode" في شاشة البدء في المثال السابق. هذا يعني أن vim سيُشغّل بسلوك شبيه بسلوك محرر vi الأصلي بدلًا من السلوك المُحسن لمحرر vim. سنحتاج في هذا الفصل إلى تشغيل vim بالسلوك المُحسَّن. توجد لدينا عدّة طرق كي نفعل ذلك:

جرب تشغيل vim بدلًا من vi. إذا تم ذلك بنجاح؛ فأنشئ الأمر البديل "alias vi='vim'" وأضفه إلى ملف ".bashrc".

طريقة أُخرى هي استخدم هذا الأمر لإضافة سطر إلى ملف إعدادات vim الخاص بك:

```
echo "set nocp" >> ~/.vimrc
```

تُحزّم بعض توزيعات لينُكس vim بطرق عديدة. تُثبّت بعض التوزيعات إصدارًا مُصغّرًا من vim افتراضيًّا الذي لا يدعم إلا عددًا قليلًا من ميزات vim. فعندما تُطَبِّق الدروس القادمة، ربما تواجه بعض الميزات الناقصة. في هذه الحالة، ثبّت الإصدار الكامل من vim.

أوضاع التعديل

لنُشغّل vi مرّة أُخرى، في هذه المرّة سنُمرر اسم ملف غير موجود إلى الأمر vi. هذه هي طريقة إنشاء الملفات الجديدة في vi.

```
[me@linuxbox ~]$ rm -f foo.txt
[me@linuxbox ~]$ vi foo.txt
```

ستحصل على الشاشة الآتية إذا جرى كل شيء على ما يرام:

```
~
```

```
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"foo.txt" [New File]
```

يشير رمز المدّة "~" في أول السطر إلى عدم وجود أي نص في ذاك السطر. هذا يظهر لنا أن الملف فارغ.
لا تضغط على أي شيء بعد!

الشيء الآخر الذي يجب علينا معرفته (عدا كيفية الخروج من البرنامج) هو أن vi يُدعى modal editor. يبدأ vi في وضع الأوامر (command mode). تُمثل جميع الأزرار تقريبًا أوامر، لذا، لو بدأنا بالكتابة مباشرةً فسيُجن vi ويُحدِث فوضى عارمة!

التبديل إلى وضع الإدخال

يجب علينا أولًا التبديل إلى وضع الإدخال (insert mode) لكي نضيف بعض النص إلى الملف. للقيام بذلك، نضغط على زر "i". ثم بعد ذلك، يجب أن نشاهد السطر الآتي في أسفل الشاشة إذا شُغِّلَ vim في الوضع المُحسّن (لن يظهر السطر في حال شُغِّلَ vim في وضع التوافقية مع vi):

```
-- INSERT --
```

نستطيع الآن إدخال بعض النص. لنجرب هذا:

```
The quick brown fox jumped over the lazy dog.
```

للخروج من وضع الإدخال والرجوع إلى وضع الأوامر؛ نضغط على زر Esc.

حفظ الملف

لحفظ التعديلات التي قمنا بها إلى الملف، يجب علينا أن ندخل أمر ex (ex command) بينما نحن في وضع الأوامر. يمكن القيام بذلك بسهولة بالضغط على زر ":". فسيظهر رمز النقطتَين الرأسيتَين في أسفل الشاشة:

```
:
```

لكتابة الملف المعدل، فإننا نُتبع النقطتين بالحرف "w" ومن ثم نضغط على Enter:

```
:w
```

سيُكتب الملف إلى القرص الصلب وستظهر رسالة تأكيد في أسفل الشاشة كالرسالة الآتية:

```
"foo.txt" [New] 1L, 46C written
```

تلميحة: إذا قرأتَ التوثيق الخاص بمحرر vim فستلاحظ تسمية (وبشكل مربك) وضع الأوامر بالوضع العادي normal mode و أوامر ex باسم command mode. لذا، كن حذرًا.

تحريك المؤشر

بينما أنت في وضع الأوامر، تستطيع استخدام المجموعة الكبيرة التي يوفرها vi من أوامر التحريك، تشترك بعض تلك الأوامر مع less. يحتوي الجدول الآتي على قائمة فرعية منها:

الجدول 12-1: أوامر تحريك المؤشر

الزر	يُحرّك المؤشر
l أو السهم الأيمن	محرف واحد إلى اليمين.
h أو السهم الأيسر	محرف واحد إلى اليسار.
j أو السهم السفلي	سطر واحد للأسفل.

k أو السهم العلوي	سطر واحد للأعلى.
0 (صفر)	إلى بداية السطر الحالي.
^	إلى أول حرف ليس فراغًا في السطر الحالي.
$	إلى نهاية السطر الحالي.
w	إلى بداية الكلمة (أو علامة الترقيم) التالية.
W	إلى بداية الكلمة التالية مع تجاهل علامات الترقيم.
b	إلى بداية الكلمة (أو علامة الترقيم) السابقة.
B	إلى بداية الكلمة السابقة مع تجاهل علامات الترقيم.
Ctrl-f أو Page Down	صفحة واحدة إلى الأسفل.
Ctrl-b أو Page up	صفحة واحدة إلى الأعلى.
numberG	إلى السطر ذي الرقم number. على سبيل المثال، 1G سينتقل إلى السطر الأول.
G	إلى آخر سطر من الملف.

لماذا تُستخدم الأوامر h و j و k و l لتحريك المؤشر؟ لأنه عندما كُتِبَ vi الأصلي، لم يكن لجميع الطرفيات أزرار الأسهم، ولأن المستخدم الماهر لا يريد أن يحرك أصابعه من لوحة المفاتيح.

يمكن إسباق العديد من الأوامر برقم، كما في أمر "G" الذي ذُكر سابقًا. بتحديد رقم قبل الأمر؛ فإننا نُحدد عدد المرات الواجب تكرار ذاك الأمر فيها. فعلى سبيل المثال، الأمر "5j" سيجعل vi يحرك المؤشر خمسة أسطر للأسفل.

التعديلات الأساسية

يحتوي أبسط أنواع التعديل على بعض الأوامر الأساسية كإدراج النص وحذفه وتحريك المؤشر والقص واللصق. بالطبع يدعم vi جميع تلك الأوامر لكن بطريقته الخاصة. ويدعم vi أيضًا التراجع (undo) لكن دعمًا محدودًا. إذا ضغطت الزر "u" في وضع الأوامر، فسيتراجع vi عن آخر تعديل قُمتَ به.

إلحاق النصوص

يدعم vi عدة طرق للدخول إلى وضع التعديل. استخدمنا مسبقًا الخيار "i" لإضافة نص.

لنعد الآن إلى ملف foo.txt:

```
The quick brown fox jumped over the lazy dog.
```

إذا أردنا إضافة نص في نهاية الجملة السابقة، فإننا نكتشف أن الأمر i لا يدعم ذلك، وذلك لأننا لا نستطيع تحريك المؤشر إلى ما بعد نهاية السطر. يوفر vi أمرًا لإضافة النص يدعى "a". إذا حركنا المؤشر إلى نهاية السطر وضغطنا على "a" فسيتجاوز المؤشر نهاية السطر ويدخل vi في وضع التعديل. وهذا ما يسمح لنا بإضافة المزيد من النص:

```
The quick brown fox jumped over the lazy dog.  It was cool.
```

لا تنسَ أن تضغط على الزر Esc للخروج من وضع التعديل.

ولأننا نريد غالبًا إضافة النص في آخر السطر، فيوفر vi اختصارًا لذلك؛ ألا وهو الأمر "A". دعنا نجربه ونُضِف بعض الأسطر إلى الملف الخاص بنا.

أولًا، سنحرّك المؤشر إلى بداية السطر باستخدام الأمر "0" (الرقم صفر). ومن ثم نضغط على "A" وندخل الأسطر الآتية:

```
The quick brown fox jumped over the lazy dog.  It was cool.
Line 2
Line 3
Line 4
Line 5
```

مرّة أُخرى، لا تنس الضغط على Esc للخروج من وضع التعديل.

كما لاحظنا، فإن الأمر "A" مفيد؛ حيث يُحرّك المؤشر إلى آخر السطر قبل بدء عملية الإدخال.

افتتاح سطر

طريقة أُخرى تسمح لنا بإدراج النص هي "بافتتاح" (opening) سطر جديد. يضيف هذا الأمر سطرًا جديدًا بين سطرين موجودين مسبقًا ويُدخِل المحرر في وضع الإدخال. يوجد أمرين مختلفين هما:

الجدول 12-2: الأوامر التي تُستخدم لافتتاح سطر

الأمر	المعنى
o	افتتاح السطر الموجود تحت السطر الحالي.
O	افتتاح السطر الموجود فوق السطر الحالي.

لإزالة الغموض، جرّب وضع المؤشر على السطر الثالث ومن ثم اضغط على زر o:

```
The quick brown fox jumped over the lazy dog.  It was cool.
Line 2
Line 3

Line 4
Line 5
```

أُفتُتِحَ سطرٌ جديد تحت السطر الثالث ودخل المحرر في وضع الإدخال. اخرج من وضع الإدخال بالضغط على Esc ومن ثم اضغط على الزر u للتراجع عن التغييرات التي قُمتَ بها.

اضغط الزر O لافتتاح السطر الذي يسبق السطر الموجود فيه المؤشر:

```
The quick brown fox jumped over the lazy dog.  It was cool.
Line 2

Line 3
Line 4
Line 5
```

اخرج من وضع الإدخال (Esc) وتراجع عن التغيرات (u).

حذف النص

كما توقعتَ، يوفر vi العديد من الطرق لحذف النصوص، تتألف جميع تلك الطرق من زر واحد أو اثنين. أولًا، يحذف الزر x الحرف الموجود في موضع المؤشر. ويمكن أن يُسبق الأمر x برقم يُحدد عدد الحروف التي ستُحذف.

الأمر d شامل أكثر. وكما في الأمر x، يمكن إسباقه بعدد يحدد عدد مرات تنفيذ أمر الحذف. بالإضافة إلى وجوب أن يُلحَق بالأمر d رقمٌ يحدد حجم النص الذي سيُحذف. يحتوي الجدول الآتي على بعض الأمثلة:

الجدول 12-3: أوامر حذف النص

الأمر	يحذف
x	الحرف الحالي.
3x	الحرف الحالي والحرفَين التاليَين.
dd	السطر الحالي.
5dd	السطر الحالي والأربعة الأسطر التالية.
dW	من موضع المؤشر الحالي إلى بداية الكلمة التالية.
d$	من موضع المؤشر الحالي إلى نهاية السطر.
d0	من موضع المؤشر الحالي إلى بداية السطر.
d^	من موضع المؤشر الحالي إلى أول محرف ليس فراغًا في السطر.
dG	من السطر الحالي حتى نهاية الملف.
d20G	من السطر الحالي حتى السطر العشرين من الملف.

ضع المؤشر عند الكلمة "it" في الملف السابق، واضغط على x ضغطًا متكررًا حتى يحذف باقي الجملة. اضغط الآن على زر u بشكل متكرر حتى تتراجع عن الحذف.

ملاحظة: يدعم vi الأصلي التراجع لمرة واحدة فقط، بينما يدعم vim عددًا أكبر من ذلك.

لنجرب الحذف مرة أخرى، لكن هذه المرة باستخدام الأمر d. ضع المؤشر على الكلمة "it" واضغط على dW لحذف الكلمة:

```
The quick brown fox jumped over the lazy dog. was cool.
Line 2
Line 3
Line 4
Line 5
```

اضغط على $d للحذف من موضع المؤشر إلى نهاية السطر:

```
The quick brown fox jumped over the lazy dog.
Line 2
Line 3
Line 4
Line 5
```

اضغط على dG للحذف من السطر الحالي حتى نهاية الملف:

```
~
~
~
~
~
```

اضغط على u ثلاث مرات للتراجع عن الحذف.

قص ونسخ ولصق النصوص

الأمر d لا يحذف النص بل يقصه أيضًا. سيُنقَل النص المحذوف إلى ما يُشبه الحافظة في كل مرة يُستخدم فيها الأمر d. ومن ثم نستطيع لصق محتويات تلك الحافظة بعد المؤشر بالأمر p أو قبل المؤشر بالأمر P.

يستخدم الأمر y لنسخ (تذكر المصطلح "yank" الذي استخدمناه سابقًا) النص بنفس الآلية التي يُستخدم فيها الأمر d لقص النص. هذه أمثلة عن استخدام الأمر y مع مختلف أوامر حركة المؤشر:

الجدول 12-4: أوامر النسخ

الأمر	ينسخ
yy	السطر الحالي.
5yy	السطر الحالي والأسطر الأربعة التالية.
yW	من موضع المؤشر حتى بداية الكلمة التالية.
y$	من موضع المؤشر حتى نهاية السطر.
y0	من موضع المؤشر حتى بداية السطر.
y^	من موضع المؤشر الحالي إلى أول محرف ليس فراغًا في السطر.

yG	من السطر الحالي إلى نهاية الملف.
y20G	من السطر الحالي حتى السطر العشرين من الملف.

لنجرب بعض أوامر النسخ واللصق. ضع المؤشر على أول سطر من النص واطبع yy لنسخ السطر الحالي. ومن ثم حَرّك المؤشر إلى آخر سطر (G) واطبع p للصق السطر السابق تحت السطر الحالي:

```
The quick brown fox jumped over the lazy dog.  It was cool.
Line 2
Line 3
Line 4
Line 5
The quick brown fox jumped over the lazy dog.  It was cool.
```

وكما في الأمثلة السابقة، اضغط على u للتراجع عن التغيير الذي قُمتَ به. اضغط P عندما يكون المؤشر موجودًا في السطر الأخير:

```
The quick brown fox jumped over the lazy dog.  It was cool.
Line 2
Line 3
Line 4
The quick brown fox jumped over the lazy dog.  It was cool.
Line 5
```

جرب بعض أوامر y الأخرى الموجودة في الجدول السابق لكي تعتاد على سلوك الأمرين p و P. لا تنسَ أن تُعيد الملف إلى حالته السابقة عند الانتهاء من تجاربك.

ضمّ الأسطر

محرر vi ثابت على الفكرة السطرية. فليس من الممكن أن تحرك المؤشر إلى نهاية السطر وأن تحذف "محرف نهاية السطر" (\n) لكي تضم السطر إلى السطر الذي يليه. لهذا السبب، يُوفّر vi أمرًا خاصًا للقيام بذلك هو الأمر J (وليس j الذي يُستخدم لتحريك المؤشر).

جرب وضع المؤشر على السطر الثالث وكتابة الأمر "J":

```
The quick brown fox jumped over the lazy dog.  It was cool.
Line 2
```

```
Line 3 Line 4
Line 5
```

البحث والاستبدال

يملك vi القدرة على تحريك المؤشر إلى مكان معين بالاعتماد على نتائج البحث. يمكنه القيام بالبحث في سطر واحد أو في كامل الملف. ويستطيع أيضًا أن يقوم بإعلام المستخدم بعمليات الاستبدال أو لا يقوم بإعلامه بذلك.

البحث في سطر واحد

يبحث الأمر f في سطر واحد ويحرّك المؤشر إلى المطابقة التالية للمحرف المحدد. على سبيل المثال، سيُحرِّك الأمر fa المؤشر إلى المطابقة التالية للحرف "a" في السطر الحالي. بعد القيام بعملية البحث عن المحارف، يمكن تكرار نفس العملية بالضغط على زر الفاصلة المنقوطة ";".

البحث في كامل الملف

يُستخدم الأمر / لتحريك المؤشر إلى المطابقة التالية لكلمة أو عبارة. يعمل هذا الأمر بما يشبه الطريقة التي يعمل فيها في برنامج less. عندما تطبع الأمر /؛ فسيظهر الرمز "/" في أسفل الشاشة. ثم أدخل الكلمة أو الجملة التي تريد البحث عنها ومن ثم اضغط على Enter. سيتحرك المؤشر إلى موضع المطابقة في النص، يمكن تكرار البحث للحصول على المطابقات التالية بالأمر n، مثال:

```
The quick brown fox jumped over the lazy dog.  It was cool.
Line 2
Line 3
Line 4
Line 5
```

ضع المؤشر في بداية السطر واطبع:

```
/Line
```

ومن ثم اضغط على Enter. سيتحرك المؤشر إلى السطر الثاني. الآن، اضغط على n وسيتحرك المؤشر إلى السطر الثالث. سيؤدي تكرار الأمر n إلى تحريك المؤشر إلى مكان المطابقة التالية إلى أن لا يبقى هنالك أيّة مُطابقات في الملف. بينما نستطيع استخدام الكلمات والجمل مع خاصية البحث، إلا أن محرر vi يدعم

استخدام التعابير النظامية (regular expressions) أيضًا. التي تمثل طريقةً قويةً جدًا في مطابقة الأنماط النصية المعقدة. سنشرح التعابير النظامية بالتفصيل في فصلٍ لاحق.

البحث والاستبدال في كامل الملف

يستخدم `vi` الأمر ex للقيام بعمليات البحث والاستبدال (تدعى "substitution" في `vi`) في مجموعة أسطر أو كامل الملف. سنستخدم الأمر الآتي لتغيير الكلمة "`Line`" إلى "`line`" في كامل الملف:

```
:%s/Line/line/g
```

لنُقسّم الأمر السابق إلى عدّة عناصر؛ ولنشرح معنى كل عنصر:

الجدول 12-5: مثال عن الشكل العام للبحث والاستبدال

العنصر	المعنى
:	يشير رمز النقطتين الرأسيتين إلى بدء أمر ex.
%	تحديد مجال الأسطر التي سيُنفّذ الأمر فيها. الرمز "%" هو اختصار يعني أن المجال هو من أول سطر حتى آخر سطر أي الملف بأكمله. يمكن أيضًا تحديد مجال الأسطر على الشكل 1,5 (لأن الملف الخاص بنا يحتوي على خمسة أسطر فقط)، أو `1,$` الذي يعني "من السطر الأول حتى آخر سطر في الملف". إذا لم يحدد المجال، فسيُنفّذ الأمر على السطر الحالي فقط.
s	تحديد العملية، في حالتنا هذه هي "substitution" أي البحث والاستبدال.
`/Line/line/`	نمط البحث والاستبدال.
g	هذا يعني أن العملية عامة (global). هذا يعني أنه ستُستبدل جميع مطابقات البحث في كل سطر. ستُستبدل أول مطابقة في كل سطر إذا حُذف هذا الخيار.

سيتغير محتوى الملف بعد تنفيذ الأمر السابق، ليصبح كالآتي:

```
The quick brown fox jumped over the lazy dog.  It was cool.
line 2
line 3
line 4
line 5
```

يمكننا أيضًا الطلب من المستخدم التأكيد على عمليات البحث والاستبدال قبل إجراءها. يتم ذلك بإضافة "c" في آخر الأمر، على سبيل المثال:

```
:%s/line/Line/gc
```

سيعيد الأمر السابق الملف إلى حالته الأصلية؛ لكن ستُسأل قبل تنفيذ أيّة عملية استبدال، وذلك بإظهار الرسالة الآتية:

```
replace with Line (y/n/a/q/l/^E/^Y)?
```

يمكن استخدام أي حرف من الأحرف الموجودة بين قوسَين كخيار. يشرح الجدول الآتي معاني تلك الأحرف:

الجدول 12-6: أزرار الموافقة على الاستبدال

الزر	المعنى
y	تأكيد عملية الاستبدال.
n	تجاوز هذه المطابقة.
a	إجراء عملية الاستبدال على جميع مطابقات النمط.
Q أو Esc	الخروج من وضع الاستبدال.
l	القيام بعملية الاستبدال الحالية ومن ثم الخروج من وضع الاستبدال (اختصار للكلمة "last").
Ctrl-e, Ctrl-y	التمرير (scroll) إلى الأسفل وإلى الأعلى على الترتيب. هذا الأمر مفيد لمشاهدة المحتوى الذي تمت مطابقته.

إذا ضغطت على y فستتم عملية الاستبدال، أما إذا ضغطت n فسيتجاوز vi هذه المطابقة وينتقل إلى المطابقة التالية.

تعديل عدّة ملفات

يكون عادةً من المفيد أن يُعّدَل أكثر من ملف في آن واحد. ربما تريد أن تجري تعديلات على أكثر من ملف أو تريد نسخ محتوى من ملفٍ إلى آخر. نستطيع، في محرر vi، فتح عدّة ملفات للتعديل بتحديدها كوسائط في سطر الأوامر:

```
vi file1 file2 file3...
```

لنخرج من جلسة vi الحالية ولننشئ ملفًا جديدًا للتعديل. اطبع الأمر wq: للخروج من vi وحفظ النص المعدل. لنُنشئ الآن ملفًا جديدًا في مجلد المنزل لكي نستطيع التجربة عليه. سنُنشئه باستخدام ناتج الأمر ls:

```
[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt
```

لنحرر الملف القديم والملف الجديد:

```
[me@linuxbox ~]$ vi foo.txt ls-output.txt
```

سيبدأ vi وسنشاهد الملف الأول على الشاشة:

```
The quick brown fox jumped over the lazy dog.  It was cool.
Line 2
Line 3
Line 4
Line 5
```

التبديل بين الملفات

للتبديل إلى الملف التالي، استخدم أمر ex الآتي:

```
:n
```

للرجوع إلى الملف السابق، استخدم:

```
:N
```

على الرغم من أننا نستطيع الانتقال من ملفٍ إلى آخر، إلا أنّ vi يفرض سياسة تمنعنا من تبديل الملفات إذا كان الملف الحالي يحتوي على أيّة تعديلات غير محفوظة. أضف علامة التعجب "!" إلى نهاية الأمر لإجبار vi على الانتقال بين الملفات.

بالإضافة إلى التبديل بين الملفات بالطريقة السابقة، يدعم vim (وبعض نسخ vi) بعض أوامر ex لتسهيل إدارة الملفات. يمكننا عرض قائمة بالملفات التي يجري تعديلها بالأمر ":buffers". سيعرض ذاك الأمر القائمة الآتية في أسفل الشاشة:

```
:buffers
   1 %a     "foo.txt"                              line 1
   2        "ls-output.txt"                        line 0
Press Enter or type command to continue
```

للتبديل إلى ملفٍ آخر، اطبع الأمر buffer: يلحقه رقم الملف الذي تريد تعديله. على سبيل المثال، للانتقال من الملف 1 (foo.txt) إلى الملف 2 (ls-output.txt) نطبع الأمر:

```
:buffer 2
```

وسيظهر الملف الثاني على شاشتنا.

فتح المزيد من الملفات للتعديل

من الممكن أيضًا إضافة المزيد من الملفات لتعديلها في جلستنا الحالية. وذلك باستخدام الأمر e: (اختصار للكلمة "edit") متبوعًا باسم الملف. لننهِ جلسة vi الحالية ونعود إلى سطر الأوامر. ومن ثم سنبدأ vi لكن هذه المرة بملفٍ واحدٍ فقط:

```
[me@linuxbox ~]$ vi foo.txt
```

ولإضافة الملف الثاني ندخل الأمر:

```
:e ls-output.txt
```

وسيظهر محتوى الملف الثاني على الشاشة. يمكننا التأكد من أن الملف الأول ما زال مفتوحًا بالأمر ":buffers":

```
:buffers
   1 #      "foo.txt"                             line 1
   2 %a     "ls-output.txt"                       line 0
Press Enter or type command to continue
```

ملاحظة: لا يمكن التبديل إلى ملفٍ مفتوح بالأمر e: باستخدام الأمرين n: أو N: يجب عليك استخدام الأمر buffer: متبوعًا برقم الملف للتبديل بين الملفات.

نسخ المحتوى من ملفٍ إلى آخر

تريد غالبًا أن تنسخ بعض محتويات أحد الملفات إلى ملفٍ آخر عندما تفتح أكثر من ملف للتعديل. يمكن القيام بذلك بسهولة بالنسخ واللصق كما تعلمنا في الفقرات السابقة. سنفتح أولًا الملفَين وننتقل إلى ملف foo.txt بطبّاعة الأمر:

```
:buffer 1
```

سيظهر الآتي على الشاشة:

```
The quick brown fox jumped over the lazy dog. It was cool.
Line 2
Line 3
Line 4
Line 5
```

من ثم سنحرك المؤشر إلى السطر الأول ونطبع الأمر yy لنسخ السطر.

سنحوّل الآن إلى الملف الثاني بطباعة:

```
:buffer 2
```

ستظهر شاشة تحتوي على معلومات بعض الملفات شبيهة بالشاشة الآتية (عُرِضَ جزء من الملف هنا فقط):

```
total 343700
-rwxr-xr-x 1  root  root    31316  2007-12-05  08:58 [
-rwxr-xr-x 1  root  root     8240  2007-12-09  13:39 411toppm
-rwxr-xr-x 1  root  root   111276  2008-01-31  13:36 a2p
-rwxr-xr-x 1  root  root    25368  2006-10-06  20:16 a52dec
-rwxr-xr-x 1  root  root    11532  2007-05-04  17:43 aafire
-rwxr-xr-x 1  root  root     7292  2007-05-04  17:43 aainfo
```

حرّك المؤشر إلى السطر الأول والصق النص الذي نسخناه من الملف السابق بالأمر "p":

```
total 325608
The quick brown fox jumped over the lazy dog. It was cool.
-rwxr-xr-x 1  root  root    31316  2007-12-05  08:58 [
-rwxr-xr-x 1  root  root     8240  2007-12-09  13:39 411toppm
```

```
-rwxr-xr-x 1  root  root   111276  2008-01-31  13:36 a2p
-rwxr-xr-x 1  root  root    25368  2006-10-06  20:16 a52dec
-rwxr-xr-x 1  root  root    11532  2007-05-04  17:43 aafire
-rwxr-xr-x 1  root  root     7292  2007-05-04  17:43 aainfo
```

إدراج ملف كامل داخل ملف آخر

من الممكن إدراج كامل محتويات ملفٍ ما في الملف الذي نعدّل عليه حاليًا. لكي نجرب ذلك، سنُنهي جلسة vi الحالية ونُنشئ واحدة جديدة بملف مفتوح واحد فقط:

```
[me@linuxbox ~]$ vi ls-output.txt
```

سنشاهد ملفنا مرّة أخرى:

```
total 325608
-rwxr-xr-x 1  root  root    31316  2007-12-05  08:58 [
-rwxr-xr-x 1  root  root     8240  2007-12-09  13:39 411toppm
-rwxr-xr-x 1  root  root   111276  2008-01-31  13:36 a2p
-rwxr-xr-x 1  root  root    25368  2006-10-06  20:16 a52dec
-rwxr-xr-x 1  root  root    11532  2007-05-04  17:43 aafire
-rwxr-xr-x 1  root  root     7292  2007-05-04  17:43 aainfo
```

حرّك المؤشر إلى السطر الثالث ومن ثم أدخل أمر ex الآتي:

```
:r foo.txt
```

يُدرج الأمر r: (اختصار للكلمة "read") ملفًا محددًا قبل موضع المؤشر. ستحتوي شاشتنا الآن على الآتي:

```
total 325608
-rwxr-xr-x 1  root  root    31316  2007-12-05  08:58 [
-rwxr-xr-x 1  root  root     8240  2007-12-09  13:39 411toppm
The quick brown fox jumped over the lazy dog. It was cool.
Line 2
Line 3
Line 4
Line 5
-rwxr-xr-x 1  root  root   111276  2008-01-31  13:36 a2p
```

```
-rwxr-xr-x 1  root  root    25368  2006-10-06  20:16 a52dec
-rwxr-xr-x 1  root  root    11532  2007-05-04  17:43 aafire
-rwxr-xr-x 1  root  root     7292  2007-05-04  17:43 aainfo
```

حفظ الملفات

توجد عدّة طرق لحفظ الملفات التي عدّلناها (كما كل شيء آخر في vi). لقد شرحنا سابقًا الأمر w:، لكن توجد عدّة طرق لحفظ الملف قد تجد بعضها مُفيدًا.

ستؤدي طباعة ZZ في وضع الأوامر إلى حفظ الملف وإنهاء vi. كذلك أمر wq: الذي يدمج بين الأمرين w: و q: الذي يؤدي أيضًا إلى حفظ الملف والخروج من vi.

يمكن تحديد وسيط اختياري للأمر w: لتحديد اسم الملف الذي سيُحفظ. أي أنه يعني "حفظ باسم...". على سبيل المثال، إذا كنا نعدل الملف foo.txt وأردنا حفظ نسخة أُخرى تسمى foo1.txt؛ فندخل الأمر الآتي:

```
:w foo1.txt
```

ملاحظة: بينما يحفظ الأمر السابق الملف باسم مختلف، إلا أنه لا يغير اسم الملف الذي تُعدّله الآن. أي بإكمالك التعديل، فإنك ستغير الملف foo.txt وليس foo1.txt.

الخلاصة

بعد تعلمنا لمهارات التعديل الأساسية؛ أصبح بإمكاننا تعديل الملفات النصية لصيانة أنظمة لينُكس. استخدام محرر vim في تنفيذ المهام الاعتيادية سيؤتي أُكُلَه. ولما كانت المحررات التي تُشبِه vi متأصلة في يونكس؛ فسنشاهد العديد من البرامج التي تأثرت بتصميمه، برنامج less هو مثالٌ عن ذلك.

الفصل الثالث عشر:

تخصيص المِحث

سنشرح في هذا الفصل أحد التفاصيل "التافهة": مِحث الصدفة! لكن سيكشف الشرح آلية العمل الداخلية للصدفة ولمحاكي الطرفية نفسه.

كما هو الحال مع العديد من الأشياء في لينُكس، مِحث الصدفة قابل للتخصيص كثيرًا، وعلى الرغم من أننا اعتبرنا المِحث مجرد أمر مُسلّم به دون أهمية، إلا أنه قد يصبح مفيدًا جدًا إذا تعلمنا طريقة التحكم فيه.

بنُية المِحث

المِحث الافتراضي لنا يشبه المِحث الآتي:

```
[me@linuxbox ~]$
```

لاحظ أنه يحتوي على اسم المستخدم واسم المُضيف ومجلد العمل الحالي، لكن كيف تم تشكيله بهذه الطريقة؟ بكل بساطة لأن المِحث يعرّف بمتغير بيئة يدعى PS1 (اختصار للعبارة "prompt string one"). نستطيع مشاهدة محتوى المتغير PS1 باستخدام الأمر echo:

```
[me@linuxbox ~]$ echo $PS1
[\u@\h \W]\$
```

ملاحظة: لا تقلق إن لم تكن النتائج عندك مطابقة للمثال السابق. تركّب كل توزيعة المِحث بشكل مختلف قليلًا (وبعضها غريب جدًا!).

من النتائج، لاحظنا أن المتغير PS1 يحتوي على عدد من المحارف كالأقواس وإشارة "@" وإشارة الدولار، لكن المحارف الباقية غامضة. يتذكر البعض منّا ورود مثل هذه الرموز في الفصل السابع عندما أطلقنا عليهم اسم المحارف الخاصة المهربة باستخدام الشرطة المائلة الخلفية (backslash-escaped special characters). هذه القائمة تحتوي على أغلب المحارف التي تعاملها الصدفة معاملةً خاصةً في العبارة المُكوّنة للمِحث:

الجدول 13-1: الأكواد الخاصة المُستخدَمة في مِحث الصدفة

الرمز	المعنى
\a	الجرس. يؤدي هذا الرمز عند وروده إلى أن يصدر الحاسوب صوتًا.
\d	التاريخ بصيغة "اليوم الشهر رقم اليوم". على سبيل المثال "Mon May 26".
\h	اسم المضيف المحلي بدون اسم النطاق.
\H	اسم المضيف المحلي كاملًا.
\j	عدد المهمات التي تُنفّذ في جلسة الصدفة الحالية.
\l	عدد أجهزة الطرفية الموصولة إلى الجهاز الحالي.
\n	محرف السطر الجديد.
\r	محرف العودة إلى بداية السطر.
\s	اسم برنامج الصدفة.
\t	الوقت الحالي بصيغة 24 ساعة كالتالي: hours:minutes:seconds.
\T	الوقت الحالي بصيغة 12 ساعة.
\@	الوقت الحالي بصيغة 12 ساعة مع إضافة AM/PM.
\A	الوقت الحالي بصيغة 24 ساعة على الشكل hours:minutes.
\u	اسم المستخدم الحالي.
\v	رقم نسخة (version) الصدفة.
\V	رقم نسخة (version) وإصدارة (release) الصدفة.
\w	مسار مجلد العمل الحالي.
\W	آخر قسم من مسار مجلد العمل الحالي (اسم المجلد فقط).
\!	رقم سجل التأريخ للأمر الحالي.

\#	عدد الأوامر التي أُدخلت في جلسة الصدفة.
\$	إظهار الرمز "$" إلا إذا كانت لديك امتيازات الجذر فعندها سيظهر الرمز "#" بدلًا عنه.
\[	يشير إلى بداية سلسلة من رمز غير مطبوع واحد أو أكثر التي تقوم بمعالجة الطرفية بطريقة أو بأخرى، كتحريك المؤشر أو تغيير ألوان النص ...إلخ.
\]	يشير إلى نهاية سلسلة الرموز غير المطبوعة.

تجربة بعض تصميمات المِحثات الأخرى

نستطيع الآن، باستخدام القائمة السابقة، تغيير المِحث. سنأخذ أولًا نسخةً احتياطيةً من محتوى المتغير PS1 لاستعادتها لاحقًا. للقيام بذلك، سوف نسند قيمة المتغير إلى متغير جديد قمنا نحن بإنشائه:

```
[me@linuxbox ~]$ ps1_old="$PS1"
```

أنشأنا متغيرًا جديدًا باسم ps1_old وأسندنا قيمة المتغير ps1 إليه. باستطاعتنا التحقق من نسخ قيمة المتغير باستخدام الأمر echo:

```
[me@linuxbox ~]$ echo $ps1_old
[\u@\h \W]\$
```

بإمكاننا استعادة المِحث الافتراضي في أي وقت خلال جلسة الطرفية بالقيام بالأمر المعاكس:

```
[me@linuxbox ~]$ PS1="$ps1_old"
```

نحن جاهزون الآن للتجربة. لنجرب إسناد سلسلة نصية فارغة:

```
[me@linuxbox ~]$ PS1=
```

إذا أسندنا لا شيء إلى المتغير ps1 فإننا نحصل على لا شيء! لا يوجد أيّ نص يظهر في المِحث! لكن المِحث ما زال موجودًا، كما طلبنا منه. ولأن مظهر المِحث غير مريح على الإطلاق، فإننا سنغيره إلى مِحث مُصغّر:

```
PS1="\$ "
```

ذاك أفضل بكثير. على الأقل إننا نعرف ماذا نفعل. لاحظ وجود الفراغ بين علامتي الاقتباس مما يؤدي إلى إظهار فراغ بين إشارة الدولار والمؤشر عند إظهار المِحث.

لنضف جرسًا إلى المِحث:

```
$ PS1="\[\a\]\$ "
```

سنسمع الآن صوت جرس في كل مرّة يظهر فيها المِحث. ربما يكون مزعجًا، إلا أنه مفيد إذا أردنا سماع صوت تنبيهي عند انتهاء تنفيذ أمر يأخذ وقتًا طويلًا. لاحظ أننا ضمَّنَّا التعبيرَين \[و \] لأن محرف الجرس \a لا يسبب طباعة أي حرف مرئي. أي أنه لا يحرّك المؤشر. لذلك، أخبرنا الصدفة أنه محرف غير مطبوع كي تُقدّر طول المِحث تقديرًا صحيحًا.

لنجرب الآن مِحثًا يحتوي على معلومات مفيدة وهي اسم المضيف والوقت:

```
$ PS1="\A \h \$ "
17:33 linuxbox $
```

تفيد إضافة الوقت إلى المِحث في حال أردنا تتبع زمن تنفيذ بعض المهام. في النهاية، سننشئ مِحثًا يُشبه المِحث الأصلي:

```
17:37 linuxbox $ PS1="<\u@\h \W>\$ "
<me@linuxbox ~>$
```

جرّب بعض الرموز الموجودة في الجدول أعلاه، وانظر هل تستطيع إنشاء مِحث جديد وجميل!

إضافة الألوان

تستجيب أغلب محاكيات الطرفية إلى محارف غير طباعية معينة للتحكم في بعض خصائص المحارف (كاللون، وإظهار النص بخط عريض، وجعل النص يومض) وموضع المؤشر. سنشرح التحكم في موضع المؤشر في وقتٍ لاحق، وسنبدأ أولًا بالألوان.

تخبط الطرفيات

لنعد إلى الزمن القديم، عندما كانت الطرفيات تُوصَل إلى الحواسيب المركزية، كانت هنالك العديد من شركات الحواسيب تنتج أنواعًا مختلفةً من الطرفيات التي يعمل كل نوع منها بطريقته الخاصة. كانت لديهم لوحات مفاتيح خاصة وطرق تفسير مختلفة لأكواد التحكم. كان لدى يونكس والأنظمة الشبيهة بِيونكس نظامَين فرعيَين معقدَين للتحكم في الطرفيات المختلفة (يُسميان termcap و terminfo). إذا بحثتَ جيدًا في إعدادات محاكي الطرفية لديك، فستجد خيارًا لتحديد نوع المحاكاة.

> في الجهود المبذولة لتوحيد "اللغة" التي تفهمها الطرفيات، طوّر المعهد القومي الأميركي للمعايير (ANSI) مجموعةً موحدةً من أكواد التحكم. يتذكر مستخدمو DOS القدامى الملف ANSI.SYS الذي كان يُستخدم لتفعيل تفسير تلك الأكواد.

يتم التحكم في اللون عادةً بإرسال "كود ANSI" إلى الطرفية مكون من سلسلة من المحارف. لا يُعرض كود التحكم على الشاشة، بل يُفسّر من قِبل الطرفية كتعليمة. وكما شاهدنا في الجدول السابق؛ يُستخدم الرمزان \[و \] لتغليف المحارف غير المطبوعة. يبدأ كود ANSI بالرقم 033 (في النظام الثماني) ويلحقه محرف خاصية (attribute character) اختياري ومن ثم التعليمة. على سبيل المثال، الكود المستخدم للون النص الأصلي هو 0، بينما اللون الأسود هو:

```
\033[0;30m
```

يحتوي الجدول الآتي على قائمة بالألوان المتاحة. لاحظ أن الألوان مقسمة إلى مجموعتين تختلفان بخاصية الخط العريض (1) التي تشير إلى الألوان "الفاتحة".

الجدول 13-2: الأكواد الخاصة التي تُستخدم لضبط ألوان النص

الرمز	اللون	الرمز	اللون
\033[0;30m	أسود.	\033[1;30m	بني غامق.
\033[0;31m	أحمر.	\033[1;31m	أحمر فاتح.
\033[0;32m	أخضر.	\033[1;32m	أخضر فاتح.
\033[0;33m	بني.	\033[1;33m	أصفر.
\033[0;34m	أزرق.	\033[1;34m	أزرق فاتح.
\033[0;35m	بنفسجي.	\033[1;35m	بنفسجي فاتح.
\033[0;36m	سماوي.	\033[1;36m	سماوي فاتح.
\033[0;37m	فضي فاتح.	\033[1;37m	أبيض.

لنجرب أن نصنع مِحثًا أحمر. سنُضيف رمز اللون في البداية:

```
<me@linuxbox ~>$ PS1="\[\033[0;31m\]<\u@\h \W>\$ "
```

```
<me@linuxbox ~>$
```

لقد نجح ذلك! لكن لاحظ أن لون جميع النص الذي سيظهر بعد المِحث أحمر. لتصحيح ذلك، نضع محرف اللون الافتراضي (0) في نهاية عبارة المِحث وذلك سيخبر الطرفية بأن تعود إلى استخدام اللون الأصلي:

```
<me@linuxbox ~>$ PS1="\[\033[0;31m\]<\u@\h \W>\$\[\033[0m\] "
<me@linuxbox ~>$
```

هذا أفضل!

من الممكن أيضًا تغيير لون خلفية النص باستخدام الأكواد الموجودة في الجدول أدناه. لا تدعم ألوان الخلفية الخاصية bold:

الجدول 13-3: الأكواد الخاصة التي تُستخدم لضبط لون الخلفية

الرمز	اللون	الرمز	اللون
\033[0;40m	أسود.	\033[0;44m	أزرق.
\033[0;41m	أحمر.	\033[0;45m	بنفسجي.
\033[0;42m	أخضر.	\033[0;46m	سماوي
\033[0;43m	بني.	\033[0;47m	فضي فاتح.

نستطيع إنشاء مِحث بخلفية حمراء بتطبيق تغيير بسيط على محرف اللون الأول:

```
<me@linuxbox ~>$ PS1="\[\033[0;41m\]<\u@\h \W>\$\[\033[0m\] "
<me@linuxbox ~>$
```

جرّب بعض أكواد الألوان وتسلّ قليلًا.

ملاحظة: عدا خاصيات العادي (0) normal و العريض (1) bold، يمكن للنص أن يعطى خاصيات إظهاره وتحته خط (4) underscore وهو يومض (5) blinking و مقلوب (7) inverse. ترفض العديد من الطرفيات إظهار النص وهو يومض لأنه مزعج للغاية!

تحريك المؤشر

توجد أكواد تُستخدم للتحكم في مكان المؤشر. تُستعمل عادةً لإظهار الساعة أو أيّة معلومة أُخرى في مكان

مختلف من الشاشة كالزاوية العليا كل مرّة يتم إظهار المِحث فيها. هذه قائمة بالأكواد التي تُستخدم لتحريك المؤشر:

الجدول 13-4: الأكواد الخاصة التي تُستخدم لتحريك المؤشر

الكود	المعنى
`\033[l;cH`	تحريك المؤشر إلى السطر l و العمود c.
`\033[nA`	تحريك المؤشر إلى الأعلى n سطر.
`\033[nB`	تحريك المؤشر إلى الأسفل n سطر.
`\033[nC`	تحريك المؤشر إلى الأمام n حرف.
`\033[nD`	تحريك المؤشر إلى الخلف n حرف.
`\033[2J`	تفريغ الشاشة وتحريك المؤشر إلى الزاوية العليا اليسرى (السطر 0 والعمود 0).
`\033[K`	مسح محتويات الشاشة من موضع المؤشر الحالي إلى نهاية السطر.
`\033[s`	حفظ مكان المؤشر الحالي.
`\033[u`	استعادة مكان المؤشر المحفوظ.

سننشئ، باستخدام الأكواد السابقة، مِحثًا يُظهر الساعة (بلون أصفر) في شريط أحمر في أعلى الشاشة في كل مرّة يظهر فيها المِحث. الكود المستخدم لإنشاء ذاك المِحث هو:

```
PS1="\[\033[s\033[0;0H\033[0;41m\033[K\033[1;33m\t\033[0m\033[u\]
<\u@\h \W>\$ "
```

لنلقِ نظرة على كل جزء من النص:

الجدول 13-5: شرح عبارة المِحث المعقدة

الجزء	المعنى
`\[`	يبدأ سلسلة محارف غير مرئية. السبب الحقيقي لها هو السماح للصدفة bash بحساب قياس المِحث الظاهر. بدونها سيوضع المؤشر في غير موضعه من قِبل ميزات تعديل سطر الأوامر.

`\033[s`	حفظ مكان المؤشر. ذلك ضروري لإعادة المؤشر إلى المكان الأصلي بعد طباعة الوقت في أعلى الشاشة. يجدر بالذكر أنه لا تدعم جميع محاكيات الطرفيات هذا الكود.
`\033[0;0H`	تحريك المؤشر إلى الزاوية العليا اليسرى، التي هي السطر 0 والعمود 0.
`\033[0;41m`	تغيير لون الخلفية إلى الأحمر.
`\033[K`	محو جميع محتويات السطر الموجود فيه المؤشر (الزاوية العليا اليسرى) ولأن لون الخلفية هو الأحمر، فستُمحى جميع محتويات السطر وتُحوّل إلى الأحمر. لاحظ أن تلك العملية لا تؤدي إلى تغيير مكان المؤشر؛ حيث سيبقى في الزاوية العليا اليسرى.
`\033[1;33m`	تحديد لون النص إلى اللون الأصفر.
`\t`	إظهار الوقت الحالي. وبينما هو عنصر "مطبوع" إلا أننا ضمَّنّاه في قسم العناصر غير المطبوعة كي لا تقوم bash بتضمين الساعة عند حساب الحجم الحقيقي للمِحث الظاهر.
`\033[0m`	إزالة اللون (النص والخلفية).
`\033[u`	استعادة مكان المؤشر الذي حُفِظ سابقًا.
`\]`	إنهاء قسم الأحرف غير المطبوعة.
`<\u@\h \W>\$`	عبارة المِحث.

حفظ المِحث

نحن لا نريد بالتأكيد أن نطبع كل هذه الرموز الهيروغليفية طوال الوقت! لذا، فإننا نحتاج إلى حفظ تلك القيمة في مكانٍ ما. يمكننا حفظ قيمة المِحث بشكل دائم بإضافتها إلى ملف "`.bashrc`". حيث نضيف السطرين الآتيين إلى الملف:

```
PS1="\[\033[s\033[0;0H\033[0;41m\033[K\033[1;33m\t\033[0m\033[u\]
<\u@\h \W>\$ "
```

```
export PS1
```

الخلاصة

صدق أو لا تصدق، توجد أشياءُ كثيرة يمكن القيام بها في المِحث بما فيها دوال وسكربتات الشِل التي لم نشرحها بعد، لكن هذه بداية جيدة. ليس الجميع مهتمًا بتغيير المِحث، لأن المِحث الافتراضي يكون عادةً مرضيًا. لكن للأشخاص الذين يريدون إضاعة وقتهم، فتوفر الصدفة ساعاتٍ من المرح لهؤلاء.

تُرِكَتْ هَذِهِ الصَفْحَةُ فَارِغَةً عَمْدًا

الباب الثالث:
المهام الشائعة والأدوات الأساسية

الفصل الرابع عشر:

إدارة الحزم

إذا قضيت بعض الوقت في مجتمعات لينُكس، فإنك ستسمع العديد من الآراء حول "أفضل" توزيعة لينُكس. غالبًا ما تصبح مثل هذه النقاشات سخيفة للغاية، وتركز على بعض الأشياء كجمالية خلفية سطح المكتب (بعض الأشخاص لا يستخدمون أوبنتو لأن لون السِمة الافتراضي هو البني!)، وبعض الأشياء الأُخرى التافهة.

أهم عوامل تحديد جودة التوزيعة هو نظام الحزم الذي تستخدمه وحجم المجتمع الداعم للتوزيعة. أثناء قضائك المزيد من الوقت في لينُكس، ستشاهد أن البرمجيات عمومًا ديناميكية وتتغير بسرعة. أغلب التوزيعات الرفيعة المستوى تطلق إصدارًا جديدًا كل ستة أشهر والعديد من البرامج تُحدث كل يوم. سنحتاج إلى أدوات جيدة لإدارة الحزم كي نستطيع مجاراة التغيرات في البرمجيات.

إدارة الحزم هي آلية تثبيت وتغيير البرمجيات في النظام. في الوقت الحالي، الحزم الموجودة في المستودعات التي يوفرها صانعو التوزيعة ترضي حاجات غالبية المستخدمين للبرامج. وهذا يختلف عما كان الحال عليه في بدايات لينُكس حيث يحتاج كل مستخدم إلى تنزيل (download) وتصريف (compile) الكود المصدري (source code) لكي يستطيع تثبيت البرمجيات. لا توجد أيّة مشكلة في البناء من المصدر؛ في الواقع، إمكانية الوصول إلى الكود المصدري للبرامج هو ميزة أساسية ومهمة من مزايا البرمجيات الحرة التي يعتمد عليها لينُكس. يعطينا (لنا، ولأي شخص آخر) القدرة على الاطلاع وتحسين النظام. لكن الحصول على البرامج مُصرّفةً وجاهزةً على شكل حزم أسرع وأسهل بالتعامل.

سنلقي في هذا الفصل نظرة على أدوات سطر الأوامر التي تستخدم لإدارة الحزم. وعلى الرغم من أن معظم التوزيعات توفر برمجيات رسومية معقدة لإدارة الحزم، لكن من الضروري أيضًا تعلم برامج سطر الأوامر، التي تستطيع القيام بالمهمات التي تعد صعبة (أو مستحيلة) بالمقارنة مع نظرائهم الرسوميين.

أنظمة التحزيم

تستخدم مختلف التوزيعات أنظمة تحزيم مختلفة. وكقاعدة عامة، الحزمة التي أُنشئت للعمل مع توزيعة معينة لن تكون متوافقة مع توزيعة ثانية. تنقسم أغلب التوزيعات إلى مخيمَين لتقنيات التحزيم: مخيم دبيان "deb." ومخيم ريدهات "rpm.". هنالك بعض الاستثناءات المثيرة للاهتمام كتوزيعة جنتو وسلاكوير و Foresight. لكن أغلب التوزيعات الأخرى تستخدم أحد نظامي الحزم السابقَين.

الجدول 14-1: عائلتَيّ نظامَيّ التحزيم الشهيرتَين

نظام التحزيم	التوزيعات (قائمة جُزئيّة)
نمط دبيان (`.deb`)	Debian, Ubuntu, Xandros, Linspire
نمط ريدهات (`.rpm`)	Fedora, CentOS, Red Hat Enterprise Linux, OpenSUSE, Mandriva, PCLinuxOS

كيف يعمل نظام الحزم

إن طريقة التوزيع المستخدمة في البرمجيات المملوكة (proprietary software) عادةً هي بيع أسطوانة تحتوي على "معالج التثبيت" لتثبيت برنامج جديد على النظام.

لا يعمل لينُكس بهذه الطريقة. جميع البرمجيات التي تتوفر لنظام لينُكس موجودة على الإنترنت. تتوفر أغلب البرامج كحزم يوفرها صانعي التوزيعة والباقي متوافر على هيئة كود مصدري قابل للبناء والتثبيت اليدوي. سنتحدث عن طريقة بناء البرنامج من المصدر في فصلٍ لاحق.

ملفات الحزم

تسمى أصغر وحدة في نظام الحزم ملف الحزمة. يكون ملف الحزمة عادةً على شكل مجموعة مضغوطة من الملفات التي تتضمن ملفات البرنامج. يمكن أن تتكون الحزمة من عدّة برامج. بالإضافة إلى الملفات التي ستُثبَّت، يحتوي ملف الحزمة على بيانات وصفية (metadata) عن الحزمة تمثل نصًا توضحيًا يحتوي معلوماتٍ عن الحزمة ومحتوياتها. إضافةً إلى ذلك، قد تحتوي العديد من الحزم على سكربتات تُنفَّذ قبل أو بعد التثبيت للقيام بعمليات الضبط والتهيئة.

تُنشأ الحزم من قِبل شخص يسمى "مشرف الحزمة" (package maintainer)، عادةً (وليس دائمًا) هو شخص من الشركة الصانعة للتوزيعة. يحصل الشخص الصانع للحزمة على الكود المصدري من كاتب البرنامج (يسمى عادةً "المنبع" [upstream provider])، ثم يبنيه وينشئ البيانات الوصفية للحزمة وأيّة سكربتات تثبيت ضرورية. يطبق ذاك الشخص عادةً تغييرات على الكود الأصلي لضمان اندماج البرنامج الذي تحويه الحزمة مع باقي مكونات التوزيعة.

المستودعات

على الرغم من أن بعض المشاريع البرمجية تختار طريقة خاصة بها للتحزيم والتوزيع؛ إلا أن أغلب الحزم الموجودة حاليًا من صنع شركات التوزيعات وأطراف أُخرى مهتمة بالأمر. تكون الحزم متوافرة لمستخدمي توزيعة ما في مستودعات (repositories) مركزية التي قد تحتوي على آلاف الحزم التي قد بُني كلٌ منها لأجل تلك التوزيعة.

قد تحتوي التوزيعة على عدد من المستودعات المختلفة لتوفير الحزم لمختلف مراحل تطوير البرمجيات. على سبيل المثال، يكون هناك عادةً مستودع "اختباري" (testing) يحتوي على حزم البرامج الاختبارية التي يستخدمها المستخدمون "الشجعان" الذين يبحثون عن العِللّ (bugs) ويبلغون عنها لإصلاحها في النسخة المستقرة. وتحتوي التوزيعة أيضًا على مستودع "تطويري" (development) الذي يحتوي على الحزم التي ستُدرج في الإصدار القادم من التوزيعة.

قد تحتوي التوزيعة أيضًا على مستودعات طرف ثالث (third-party repositories) التي تُستخدم لتوفير البرامج التي لا يُسمح -لأسباب قانونية مثل DRM...- بتضمينها في التوزيعة. مثال شهير عليها هو دعم تشفير أقراص DVD الذي لا يعتبر قانونيًا في الولايات المتحدة الأميركية. تعمل مستودعات الطرف الثالث في الدول التي لا تملك مثل هذه القوانين. تكون تلك المستودعات عادةً مستقلة عن التوزيعة، لكن يجب علينا معرفة طريقة إعدادها.

الاعتماديات

نسبة قليلة جدًا من البرامج لا تعتمد على برامج أُخرى لكي تقوم بمهامها. تتشارك النشاطات الشائعة، كالدخل والخرج على سبيل المثال، بين العديد من البرامج عن طريق ما يسمى "مكتبة مشتركة" (shared library)، التي توفر خدمات مهمة لأكثر من برنامج.

إذا تطلبت حزمةٌ ما مكتبةً مشتركةً، فيُقال أن لديها "اعتمادية" (dependency). توفر نظم إدارة الحزم الحالية طريقة لحل مشاكل الاعتماديات، وذلك بالتحقق من تثبيت جميع الاعتماديات عند تثبيت حزمة ما.

الأدوات عالية المستوى ومنخفضة المستوى لإدارة الحزم

تحتوي نُظم إدارة الحزم عادةً على نوعَين من الأدوات: أدوات منخفضة المستوى التي تقوم بمهمات كتثبيت وإزالة ملفات الحزم، وأدوات عالية المستوى التي تبحث في البيانات الوصفية للحزم وتحلّ مشاكل الاعتماديات. سنلقي نظرة في هذا الفصل على الأدوات التي توفرها التوزيعات التي تعتمد على نمط دبيان (كأوبنتو وغيرها) وتلك التي توفرها التوزيعات التي تعتمد على نمط ريدهات. وعلى الرغم من أن جميع التوزيعات التي تعتمد على نمط ريدهات تستخدم الأداة منخفضة المستوى ذاتها (rpm)؛ إلا أنها تستخدم أدوات عالية المستوى مختلفة. سنشرح في نقاشنا هذا، الأداة العالية المستوى yum التي تُستخدم من قِبل توزيعة فيدورا و RHEL و CentOS. التوزيعات الأخرى التي تعتمد على نمط ريدهات توفر أدوات أخرى عالية المستوى بميزات متقاربة.

الجدول 2-14: أدوات أنظمة التحزيم

التوزيعات	الأدوات منخفضة المستوى	الأدوات عالية المستوى
نمط دبيان	`dpkg`	`apt-get, aptitude`
فيدورا، RHEL، CentOS	`rpm`	`yum`

المهمات الشائعة في إدارة الحزم

هنالك العديد من الميزات التي يمكن القيام بها باستخدام أدوات سطر الأوامر لإدارة الحزم، إلا أننا سنناقش أشهرها. يجدر بالذكر أن بعض الأدوات المنخفضة المستوى تدعم أيضًا إنشاء الحزم، لكن هذا الموضوع خارج عن نطاق هذا الكتاب.

سنستخدم في النقاش الآتي المصطلح "اسم الحزمة" للدلالة على الاسم الحقيقي للحزمة، الذي يختلف عن مصطلح "ملف الحزمة" الذي يمثل مسار الملف الذي يحتوي على الحزمة.

العثور على حزمةٍ ما في مستودع

يمكن العثور على حزمةٍ ما في مستودع بالبحث عن اسم أو وصف الحزمة في أسماء الحزم أو البيانات الوصفية التي توفرها، باستخدام الأدوات عالية المستوى.

الجدول 3-14: أوامر البحث عن الحزم

نمط التحزيم	الأوامر
دبيان	`apt-get update` `apt-cache search search_string`
ريدهات	`yum search search_string`

على سبيل المثال، للبحث في مستودع yum عن محرر emacs، نستخدم الأمر الآتي:

```
yum search emacs
```

تثبيت الحزم من المستودعات

تسمح الأدوات العالية المستوى بتنزيل حزمة ما من مستودع وتثبيتها مع جميع اعتمادياتها.

الجدول 14-4: أوامر تثبيت الحزم

نمط التحزيم	الأوامر
دبيان	`apt-get update` `apt-get install package_name`
ريدهات	`yum install package_name`

على سبيل المثال، لتثبيت محرر emacs من مستودع apt فإننا ننفذ الأمر:

```
apt-get update; apt-get install emacs
```

تثبيت حزمة من ملف حزمة

إذا نُزِّل ملف حزمة من مصدر آخر غير المستودع، فيمكن تثبيته مباشرةً (لكن دون حل مشكلة الاعتماديات) باستخدام أداة منخفضة المستوى.

الجدول 14-5: أوامر تثبيت الحزم المنخفضة المستوى

نمط التحزيم	الأوامر
دبيان	`dpkg --install package_file`
ريدهات	`rpm -i package_file`

مثلًا: إذا نُزِّل ملف الحزمة "emacs-22.1-7.fc7-i386.rpm" من مكان آخر غير المستودع، فيمكن تثبيته كالآتي:

```
rpm -i emacs-22.1-7.fc7-i386.rpm
```

ملاحظة: لما كانت هذه الطريقة تستخدم برنامج rpm المنخفض المستوى للقيام بعملية التثبيت، فإنه لن يحل مشكلة الاعتماديات. إذا وجد rpm اعتماديةً ناقصةً، فإنه سينتهي مع إظهار رسالة خطأ.

إزالة الحزم

تُزال الحزم باستخدام الأدوات عالية المستوى أو منخفضة المستوى. يُظهر الجدول الآتي طريقة استخدام الأدوات عالية المستوى:

الجدول 14-6: أوامر إزالة الحزم

نمط التحزيم	الأوامر
دبيان	`apt-get remove package_name`
ريدهات	`yum erase package_name`

مثلًا: لإزالة حزمة emacs من نظام مبني على دبيان:

```
apt-get remove emacs
```

تحديث الحزم من المستودعات

إحدى أشهر مهام إدارة الحزم هي إبقاء النظام محدّثًا تحديثًا مستمرًا (أي استخدام آخر إصدارات الحزم). تُنفِّذ الأدوات العالية المستوى هذا الأمر المهم في خطوة واحدة فقط:

الجدول 14-7: أوامر تحديث الحزم

نمط التحزيم	الأوامر
دبيان	`apt-get update; apt-get upgrade`
ريدهات	`yum update`

مثال: لتطبيق جميع تحديثات الحزم المتوفرة في نظام دبيان:

```
apt-get update; apt-get upgrade
```

تحديث حزمة ما من ملف الحزمة

إذا نزّلت حزمة من مصدر آخر غير المستودع، فيمكن تثبيتها مستبدلةً الإصدار الأقدم:

الجدول 14-8: أدوات التحديث المنخفضة المستوى

نمط التحزيم	الأوامر
دبيان	`dpkg --install package_file`
ريدهات	`rpm -U package_file`

على سبيل المثال: تحديث حزمة emacs من ملف الحزمة "emacs-22.1-7.fc7-i386.rpm" على نظام

ريدهات:

```
rpm -U emacs-22.1-7.fc7-i386.rpm
```

ملاحظة: لا يوفر dpkg خيارًا خاصًا لتحديث الحِزم بالمقارنة مع نظيره rpm.

عرض الحزم المُثّبتة

هذه الأوامر تُستخدم لعرض قائمة بجميع الحزم المُثَبَّتة على جهازك:

الجدول 14-9: أوامر عرض قائمة بجميع الحزم

نمط التحزيم	الأوامر
دبيان	`dpkg --list`
ريدهات	`rpm -qa`

تحديد فيما إن كانت حزمة مثبتة أم لا

الأدوات المنخفضة المستوى الآتية تظهر إذا كانت الحزمة المحددة مثبتة على النظام أم لا:

الجدول 14-10: أوامر معرفة حالة الحزم

نمط التحزيم	الأوامر
دبيان	`dpkg --status package_name`
ريدهات	`rpm -q package_name`

مثال: لتحديد فيما إن كانت حزمة emacs مثبتة على نظام مبني على دبيان:

```
dpkg --status emacs
```

إظهار معلومات حول حزمة مثبتة

في حال عرفتَ اسم الحزمة، فبإمكانك استخدام الأوامر الآتية لإظهار شرح عن عمل تلك الحزمة:

الجدول 14-11: أوامر إظهار معلومات حول الحزم

نمط التحزيم	الأوامر
دبيان	`apt-cache show package_name`
ريدهات	`yum info package_name`

مثال: عرض شرح عن الحزمة emcas في نظام مبني على دبيان:

```
apt-cache show emacs
```

معرفة أيّة حزمة ثبّتت ملفًا ما

لتحديد أية حزمة ثبَّتَت ملفًا محددًا، فإننا نستخدم الأوامر الآتية:

الجدول 14-12: أوامر التعرف على ملفات الحزم

نمط التحزيم	الأوامر
دبيان	`dpkg --search file_name`
ريدهات	`rpm -qf file_name`

مثال: لمعرفة الحزمة المسؤولة عن تثبيت الملف /usr/bin/vim في نظام ريدهات:

```
rpm -qf /usr/bin/vim
```

الخلاصة

سنستكشف في الفصول الآتية العديد من البرامج المختلفة التي تغطي طيفًا واسعًا من المهام. بينما أغلب تلك البرامج تكون مثبتةً افتراضيًا، لكنك قد تحتاج إلى تثبيت بعض الحزم الإضافية إن لم تكن تلك البرامج متوفرةً على نظامك. لكن لم يعد تثبيت وإدارة الحزم أمرًا صعبًا، بعد أن تعلمنا ذلك في هذا الفصل.

خُرافة تثبيت البرمجيات في لينُكس!

يقع الأشخاص الذين يهاجرون من منصاتٍ أُخرى في بعض الأحيان ضحيةً لخرافة أن تثبيت البرمجيات في لينُكس صعب نوعًا ما بسبب اختلاف أنظمة التحزيم بين توزيعة وأخرى. حسنًا ذاك قصور لكن فقط للبرمجيات المملوكة التي ينشر مالكوها حزمًا ثنائية فقط (أي لا يوفرون المصدر)

لبرمجياتهم السرّية!

تعتمد برمجيات لينُكس على فكرة المصدر المفتوح. إذا أصدر مطور برنامجٍ ما الكود المصدري لبرنامجه، فإن من المرجح أن يحزّم شخصٌ ما يعمل مع فريق توزيعة معينة البرنامج ويضيفه إلى المستودع. تتحقق هذه الطريقة من اندماج لينُكس مع باقي برمجيات التوزيعة بالإضافة إلى توفير مكان واحد "للتسوق" بالنسبة للمستخدم بدل بحثه عن البرمجيات في مواقعها الرسمية.

يتم التعامل مع تعريفات الأجهزة بشكل مشابه للطريقة السابقة؛ إلا أنها بدلًا من أن تكون موجودةً كأجزاء منفصلة في مستودعات التوزيعة، فإنها ستندمج مع نواة لينُكس نفسها. عمومًا، لا يوجد شيء اسمه "تعريف جهاز ما" في لينُكس. إما أن تدعم النواة الجهاز أو لا تدعمه (تدعم نواة لينُكس العديد من الأجهزة حتى أن عددها أكثر بكثير من ما يدعم ويندوز! لكن ذلك ليس مواساةً لعدم دعم لينُكس لأي جهاز). في حال كان أحد الأجهزة غير مدعوم، فيجب عليك البحث عن السبب. غالبًا ما يكون سبب عدم دعم جهاز ما هو واحد من الأسباب الآتية:

1. الجهاز حديث جدًا. لأن العديد من صانعي العتاد لا يدعمون تطوير لينُكس دعمًا نشطًا، فسيتطوع أحد المبرمجين من مجتمع لينُكس لكتابة كود النواة الذي يدعم ذاك الجهاز.
2. الجهاز غريب جدًا. لا تدعم جميع التوزيعات جميع الأجهزة المتوفرة. تبني كل توزيعة النواة الخاصة بها، ولأن النواة قابلة للتخصيص بشكل كبير (وهو الشيء الذي يسمح بتشغيل لينُكس على كل شيء من الساعات وحتى الحواسيب الخارقة!)، فربما تجاهل صانعو التوزيعة جهازًا معينًا. وبتحديد وتنزيل الكود المصدري للتعريف، فبإمكانك (نعم، أنت) بناء وتثبيت التعريف بنفسك. تلك المهمة ليست صعبة كما تظن. سنتحدث عن طريقة بناء التطبيقات في فصلٍ لاحق.
3. يخفي صانع الجهاز شيئًا ما. حيث لم ينشر الكود المصدري للتعريف، ولم يصدر حتى التوثيق التقني لكي يُنشئ أحد الأشخاص التعريف له. وهذا يعني أنه يريد إبقاء الواجهة البرمجية للجهاز سرّية. ولأننا لا نريد أي أسرار في حاسوبنا، فإني أنصحك (وبشدة) أن تنتزع ذاك الجهاز وتلقيه في سلّة المهملات التي بجوارك مع باقي الأشياء غير المفيدة!

الفصل الخامس عشر:

أجهزة التخزين

عالجنا، في الفصول السابقة، البيانات على مستوى الملف. سنتعامل، في هذا الفصل، مع مستوى الأقراص. لدى لينُكس إمكانيات هائلة فيما يتعلق بالتعامل مع أجهزة التخزين، باختلاف أنواعها. مثل أجهزة التخزين الفيزيائية كالقرص الصلب أو التخزين عبر الشبكة أو أجهزة تخزين وهمية مثل RAID (اختصار للعبارة "Redundant Array of Independent Disks") أو LVM (اختصار للعبارة "Logical Volume Manager").

لكن، ولأن هذا الكتاب ليس عن إدارة الأنظمة، لن نشرح كل موضوع من تلك المواضيع شرحًا معمّقًا. ما سنحاول تقديمه هو التعرف على بعض الأدوات والأوامر المهمة التي تُستخدم لإدارة أجهزة التخزين.

سنحتاج، للقيام بالتمارين في هذا الفصل، إلى قرص USB، وقرص CD-RW (للأجهزة التي توفر ناسخة CD-ROM) وقرص مرن (أيضًا للأجهزة التي توفر ذاك الجهاز).

سنلقي نظرة على الأوامر الآتية:

- mount – وصل نظام الملفات.
- umount – فصل نظام الملفات.
- fsck – تفحص وإصلاح نظام الملفات.
- fdisk – تعديل جدول الأقسام (partition table).
- mkfs – إنشاء نظام ملفات.
- fdformat – تهيئة قرص مرن.
- dd – كتابة كتل (blocks) من البيانات مباشرةً إلى قرص.
- genisoimage (mkiofs) – إنشاء صورة قرص بصيغة ISO 9660.
- wodim (cdrecord) – كتابة البيانات إلى قرص ضوئي.
- md5sum – حساب بصمة MD5.

وصل وفصل أجهزة التخزين

النقلات النوعية التي حدثت في الآونة الأخيرة في لينُكس جعلت إدارة الأجهزة مهمةً سهلةً جدًا لِمستخدمي سطح المكتب. في أغلب الأحيان، نضيف القرص إلى نظامنا وسوف يعمل مباشرةً دون تدخل

منا. في الماضي (ربما 2004) كانت كل هذه الأشياء يجب أن تتم يدويًا. يبقى الأمر يدويًا إلى حدٍ ما في الخوادم لأنها تحتوي على قدرات تخزينية ضخمة وإعدادات كثيرة.

أول خطوة في إدارة أجهزة التخزين هي إضافة الجهاز إلى شجرة نظام الملفات في النظام. تسمح هذه العملية (تسمى الوصل mounting) باعتبار الجهاز جزءًا من نظام التشغيل. وكما نتذكر في الفصل الثاني. لدى الأنظمة الشبيهة بِيونكس، كَلينُكس، شجرة نظام ملفات وحيدة وتوصل فيها الأجهزة في نقاط مختلفة. هذا يختلف اختلافًا كاملًا مع الأنظمة الأخرى كنظامَي MS-DOS وويندوز اللذان يستخدمان شجرة نظام ملفات لكل جهاز (على سبيل المثال `\:C` `\:D` إلخ.).

يوجد هنالك ملف `etc/fstab/` الذي يحتوي قائمة بالأجهزة (عادة أقسام القرص الصلب) التي تُوصَل في وقت الإقلاع. لدينا مثلًا الملف `etc/fatab/` من توزيعة فيدورا:

```
LABEL=/12       /         ext3   defaults       1 1
LABEL=/home     /home     ext3   defaults       1 2
LABEL=/boot     /boot     ext3   defaults       1 2
tmpfs           /dev/shm  tmpfs  defaults       0 0
devpts          /dev/pts  devpts gid=5,mode=620 0 0
sysfs           /sys      sysfs  defaults       0 0
proc            /proc     proc   defaults       0 0
LABEL=SWAP-sda3 swap      swap   defaults       0 0
```

معظم أنظمة الملفات المعروضة في المثال السابق وهمية وخارجة عن نطاق نقاشنا. سنركز نقاشنا، لأغراض هذا الفصل، على أول ثلاثة عناصر:

```
LABEL=/12       /         ext3   defaults       1 1
LABEL=/home     /home     ext3   defaults       1 2
LABEL=/boot     /boot     ext3   defaults       1 2
```

تمثل الأسطر السابقة قطاعات القرص الصلب. كل سطر يحتوي على ستة حقول وهي:

الجدول 15-1: حقول الملف etc/fstab/

الحقل	المحتويات	الشرح
1	الجهاز	تقليديًا، يحتوي هذا الحقل على الاسم الحقيقي لملف الجهاز الذي يرتبط مع الجهاز الفيزيائي. على سبيل المثال، `dev/hda1/` (الجهاز الموصول على أول محطة IDE في الحاسوب). لكن ومع التطور الحاصل مع الحواسيب الحالية التي تحتوي على العديد من

		أجهزة التخزين القابلة للوصل السريع في أثناء تشغيل الحاسوب (كأقراص USB)، فإن العديد من توزيعات لينُكس الحديثة تستخدم أسماء نصية للأجهزة. يُقرأ هذا الاسم (الذي يُضاف إلى الجهاز عند تهيئته) من قِبل نظام التشغيل عندما يُضاف الجهاز إلى النظام. نستطيع التعرف على الجهاز، باختلاف اسم الملف المرتبط مع الجهاز، عن طريق هذه الآلية.
2	نقطة الوصل	المجلد الذي يُوصل إليه الجهاز في شجرة نظام الملفات.
3	نوع نظام الملفات	يسمح لينُكس بوصل العديد من أنواع أنظمة الملفات. أغلب أنظمة ملفات لينُكس هي من نوع ext3 أو ext4. لكن نظام لينُكس يدعم العديد من أنظمة الملفات الأخرى، كنظام FAT16 (msdos), FAT32 (vfat), NTFS (ntfs) CD-ROM (iso9660) ...إلخ.
4	الخيارات	يمكن وصل أنظمة الملفات بخيارات مختلفة. من الممكن على سبيل المثال، وصل أنظمة الملفات للقراءة فقط، أو منع تنفيذ جميع البرامج الموجودة فيها (وهي ميزة أمنية مهمة للأقراص القابلة للإزالة).
5	التواتر (frequency)	رقم يحدد فيما إذا كان ومتى ستُأخذ نسخة احتياطية من نظام الملفات باستخدام الأمر dump.
6	الترتيب	رقم يحدد في أي ترتيب سيُفحص نظام الملفات باستخدام الأمر fsck.

عرض قائمة بأنظمة الملفات الموصولة

يُستخدم الأمر mount لوصل أنظمة الملفات. تنفيذ الأمر بدون أي وسيط يجعله يعرض قائمة بأنظمة الملفات الموصولة حاليًا:

```
[me@linuxbox ~]$ mount
/dev/sda2 on / type ext3 (rw)
proc on /proc type proc (rw)
```

```
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sda5 on /home type ext3 (rw)
/dev/sda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
fusectl on /sys/fs/fuse/connections type fusectl (rw)
/dev/sdd1 on /media/disk type vfat (rw,nosuid,nodev,noatime,
uhelper=hal,uid=500,utf8,shortname=lower)
twin4:/musicbox on /misc/musicbox type nfs4 (rw,addr=192.168.1.4)
```

بنية الأسطر السابقة هي: "الجهاز" في "نقطة_الوصل" ذو النوع "نوع_نظام_الملفات" "الخيارات" (إن وُجِدَت). على سبيل المثال، يُظهر أول سطر أن الجهاز /dev/sda2 قد وصِلَ في جذر نظام الملفات ونوعه ext3 بنمط القراءة والكتابة ("rw"). تُظهِر القائمة أيضًا قيدين مثيرين للاهتمام في آخر القائمة. يُظهر القيد ما قبل الأخير كرت ذاكرة SD في قارئ الذواكر موصول في /media/disk. آخر قيد يُظهر قرصًا شبكيًا (موصل عبر الشبكة) موصولًا في /misc/musicbox.

كأول تجربة لنا، سنعمل مع CD-ROM. لنلقِ نظرة أولًا على النظام قبل إدراج قرص CD-ROM:

```
[me@linuxbox ~]$ mount
/dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/hda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
```

هذا هو ناتج تنفيذ أمر mount على نظام CentOS 5 الذي يستخدم LVM لإنشاء نظام الملفات الجذر. وكباقي توزيعات لينُكس الحديثة، سيحاول هذا النظام أن يقوم بوصل قرص CD-ROM تلقائيًا. سنشاهد الناتج الآتي بعد أن نضع قرص CD-ROM:

```
[me@linuxbox ~]$ mount
/dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
```

```
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/hda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
/dev/hdc on /media/live-1.0.10-8 type iso9660 (ro,noexec,nosuid,
nodev,uid=500)
```

سنشاهد، بعد وضع القرص، إضافة قيد واحد فقط ألا وهو القيد الأخير الذي يُعبّر عن القرص المضغوط (أي الجهاز dev/hdc/ في هذا النظام) وقد تم وصله إلى media/live-1.0.10-8/ ونوعه iso9660 (أي CD-ROM). لأغراض هذا الفصل، سنكون مهتمين باسم الجهاز فقط. غالبًا ما سيكون اسم الجهاز مختلفًا عندما تجرّب بنفسك.

تحذير: في الأمثلة القادمة، من المهم جدًا أن تنتبه جديًا إلى اسم الجهاز في نظامك ولا تستخدم أسماء الأجهزة في نص المثال!

لاحظ أيضًا أن أقراص "audio CD" ليست كأقراص CD-ROM، لا تحتوي أقراص audio CD على نظام ملفات ولا يمكن أن تُوصَل بالمعنى المتعارف عليه.

الآن وبعد أن عرفنا اسم جهاز قرص CD-ROM، لنفصل القرص ومن ثم نعيد وصله لكن في مكان مختلف في شجرة نظام الملفات. للقيام بذلك، علينا استخدام حساب الجذر (باستخدام الأمر المناسب لتوزيعتك) وفصل القرص بالأمر umount (لاحظ طريقة التهجئة):

```
[me@linuxbox ~]$ su -
Password:
[root@linuxbox ~]# umount /dev/hdc
```

الخطوة الآتية هي إنشاء "نقطة وصل" (mount point) للقرص. نقطة الوصل هي بكل بساطة عبارة عن مجلد في مكانٍ ما في شجرة نظام الملفات. لا شيء خاص يميز ذاك المجلد. ولا حتى أن يكون مجلدًا فارغًا! لكن إن وصلتَ جهازًا في مجلد غير فارغ، فلن تستطيع مشاهدة محتويات المجلد السابقة حتى تفصلَ الجهاز. لذا من الأفضل إنشاء مجلد جديد:

```
[root@linuxbox ~]# mkdir /mnt/cdrom
```

أخيرًا، نقوم بوصل قرص CD-ROM في نقطة الوصل الجديدة. يُستخدم الخيار t- لتحديد نوع نظام الملفات:

```
[root@linuxbox ~]# mount -t iso9660 /dev/hdc /mnt/cdrom
```

بعد ذلك، نعاين محتويات قرص CD-ROM في نقطة الوصل الجديدة:

```
[root@linuxbox ~]# cd /mnt/cdrom
[root@linuxbox cdrom]# ls
```

لاحظ ماذا سيحدث إذا حاولنا فصل القرص:

```
[root@linuxbox cdrom]# umount /dev/hdc
umount: /mnt/cdrom: device is busy
```

لماذا؟ السبب وراء ذلك هو عدم قدرتنا على فصل جهازٍ ما إذا كانت إحدى العمليات تستخدم ذاك الجهاز. في هذه الحالة، لقد حولّنا مجلد العمل الحالي إلى نقطة الوصل، مما أدى إلى جعل الجهاز مستخدمًا. يمكننا تجاوز هذه الإشكالية بسهولة بتغييرنا مجلد العمل الحالي لأي مجلد عدا نقطة الوصل:

```
[root@linuxbox cdrom]# cd
[root@linuxbox ~]# umount /dev/hdc
```

فُصِلَ الآن الجهاز بنجاح.

لماذا فصل الأجهزة مهم جدًا

إذا ألقينا نظرة على ناتج الأمر free، فسنجد العديد من الإحصائيات حول استخدام الذاكرة، وسنجد إحدى تلك الإحصائيات مُعَنونةً بالكلمة "buffers". صُمِّمَت أنظمة الحواسيب لتعمل أسرع ما يمكن. لكن إحدى معوقات سرعة النظام هي الأجهزة البطيئة. الطابعات على سبيل المثال. وحتى أسرع طابعات العالم تكون بطيئة جدًا حسب معايير الحاسوب. سيكون الحاسوب بطيئًا للغاية إذا توقف وانتظر لإنهاء طباعة ورقة. كان ذلك كارثةً حقيقيةً في الأيام الأولى للحواسيب الشخصية (قبل تعدد المهام). إذا كُنتَ تعمل على ملف نصي أو ورقة عمل، فسيتوقف الحاسوب عن الاستجابة في كل مرّة تستخدم فيها الطابعة. سيُرسِل الحاسوب البيانات إلى الطابعة في أسرع وقتٍ ممكن، لكن ذلك سيكون بطيئًا لأن الطابعات لا تطبع بسرعة. حُلَّت المشكلة باستخدام "حافظة الطابعة" (printer buffer)، التي هي جهاز يحتوي على ذاكرة RAM التي تكون وسيطًا ما بين الحاسوب والطابعة. باستخدام حافظة الطابعة، يمكن للحاسوب أن يُرسِل الملف إلى تلك الحافظة ومن ثم يكمل الحاسوب مهامه. في أثناء

ذلك، تطبع الطابعة المستندات الموجودة في الحافظة "ببطء".

فكرة استخدام الحافظات شائعة كثيرًا في الحواسيب لجعلها أسرع. حيث لا تترك مجالًا لعمليات القراءة والكتابة في الأجهزة البطيئة أن تؤثر على سرعة النظام. تُخزِّن أنظمة التشغيل البيانات التي تُقرأ من أو تُكتب إلى أجهزة التخزين في الذاكرة لأطول وقتٍ ممكن قبل بدء تعاملها مع الجهاز البطيء. في نظام لينُكس على سبيل المثال، ربما تلاحظ أن النظام يستهلك مقدارًا أكبر من الذاكرة كلما استمر في العمل. هذا لا يعني أن لينُكس "يستهلك" الذاكرة، بل أن لينُكس يستثمر الذاكرة المتوفرة لإنشاء حافظات قدر الإمكان.

يسمح استخدام الحافظات بالكتابة بسرعة إلى أجهزة التخزين، لأن الكتابة على القرص ستؤجل إلى وقتٍ لاحق بينما تكون البيانات التي من المفترض كتابتها على القرص موجودة في الذاكرة. ويكتبها النظام إلى القرص من حينٍ لآخر.

فصل جهازٍ ما يؤدي إلى كتابة جميع البيانات المتبقية في الذاكرة إلى الجهاز كي يُزال بأمان. إذا أُزيل القرص قبل فصله، فسيكون هنالك احتمال بعدم نقل جميع البيانات إلى القرص. في بعض الحالات، تحتوي تلك البيانات على تحديث مهم لشجرة المجلدات، مما يؤدي إلى عطب في نظام الملفات، أحد أسوأ الأشياء التي قد تحدث للحاسوب!

تحديد أسماء الأجهزة

من الصعب، في بعض الأحيان، أن نحدد اسم أحد الأجهزة. لم يكن ذلك صعبًا للغاية في السابق. الجهاز دائمًا في نفس المكان ولا يتغير أبدًا. الأنظمة الشبيهة بِيونكس تحب هذا الأمر. بالعودة إلى الأيام التي طُوِّرَ يونكس فيها، فإن "تغيير محرك الأقراص" يعني انتزاع جهاز بحجم آلة غسيل الملابس من غرفة الحاسوب. أما حاليًا، فإن عتاد الحاسوب أصبح أكثر ديناميكيةً وتطور لينُكس ليصبح أكثر مرونةً من أسلافه.

استخدمنا في الأمثلة السابقة قدرة أنظمة لينُكس الحديثة على وصل الأقراص تلقائيًا. لكن ماذا لو كنا ندير خادمًا ما أو أيّة بيئة أُخرى لا تقوم بذلك؟ كيف نستطيع تحديد الاسم؟

لنلقِ نظرة أولًا على طريقة تسمية النظام للأقراص. إذا عرضنا قائمة بمحتويات المجلد dev/ (مكان وجود الأجهزة)، نستطيع مشاهدة العديد من الأجهزة:

```
[me@linuxbox ~]$ ls /dev
```

تكشف محتويات المجلد dev/ وجود بعض الأنماط لتسمية الأجهزة، وهذا بعضها:

الجدول 15-2: أسماء أجهزة التخزين في لينُكس

النمط	الجهاز
/dev/fd*	أجهزة الأقراص المرنة.
/dev/hd*	أجهزة IDE في الأنظمة القديمة. اللوحات الأم التقليدية تحتوي على "قناتَيّ" وصل لأجهزة IDE كل قناة تتصل مع كبل يحتوي على نُقطتَيّ وصل للأجهزة. أول جهاز موصول بالكبل يسمى master أما ثاني جهاز يسمى slave. أسماء الأجهزة مرتبة على النحو الآتي: dev/hda/ للجهاز الأول في القناة الأولى. dev/hdb/ للجهاز الثاني في القناة الأولى. dev/hdc/ للجهاز الأول في القناة الثانية وهكذا. يشير الرقم المحلق إلى رقم القسم في الجهاز. مثلًا dev/hda1/ يشير إلى القسم الأول في القرص الأول في النظام. بينما يشير dev/hda/ إلى الجهاز بأكمله.
/dev/lp*	الطابعات.
/dev/sd*	أقراص SCSI. في أنظمة لينُكس الحديثة، تُعامِل النواة جميع الأقراص (بما فيها الأقراص الصلبة وأقراص USB ...إلخ.) كأقراص SCSI. نظام الأسماء المُتَّبَع شبيه بالنظام الذي تستخدمه الأجهزة من نوع dev/hd*/ المشروح بالأعلى.
/dev/sr*	الأجهزة البصرية (قارئات وناسخات CD/DVD).

قد يوجد أيضًا وصلات رمزية كالوصلة dev/cdrom/ أو dev/dvd/ أو dev/floppy/، التي تشير إلى ملفات الأجهزة الأصليّة.

إذا كنت تعمل على نظام لا يقوم بوصل الأقراص القابلة للإزالة تلقائيًا، باستطاعتك استخدام التقنية الآتية لتحديد إذا أُضيف قرص إلى الحاسوب. أولًا راقب الملف var/log/messages/ أو الملف var/log/syslog/ (ربما تحتاج إلى امتيازات الجذر للقيام بذلك):

```
[me@linuxbox ~]$ sudo tail -f /var/log/messages
```

ستظهر آخر الأسطر الذي يحتويها الملف. أضف الآن القرص القابل للإزالة (استخدمنا في هذا المثال قرص فلاش بسعة 16 ميغابايت) وستلاحظ النواة القرص مباشرةً وتُبلِّغ عن ذلك:

```
Jul 23 10:07:53 linuxbox kernel: usb 3-2: new full speed USB device us-
ing uhci_hcd and address 2
Jul 23 10:07:53 linuxbox kernel: usb 3-2: configuration #1 chosen from
1 choice
```

```
Jul 23 10:07:53 linuxbox kernel: scsi3 : SCSI emulation for USB Mass
Storage devices
Jul 23 10:07:58 linuxbox kernel: scsi scan: INQUIRY result too short
(5), using 36
Jul 23 10:07:58 linuxbox kernel: scsi 3:0:0:0: Direct-Access  Easy
Disk 1.00 PQ: 0 ANSI: 2
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] 31263 512-byte hard-
ware sectors (16 MB)
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Write Protect is off
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Assuming drive
cache: write through
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] 31263 512-byte hard-
ware sectors (16 MB)
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Write Protect is off
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Assuming drive
cache: write through
Jul 23 10:07:59 linuxbox kernel: sdb: sdb1
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Attached SCSI remov-
able disk
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: Attached scsi generic sg3
type 0
```

الآن اضغط على `Ctrl-c` للعودة إلى المِحث. الأجزاء التي تهمنا من المخرجات هي تلك التي تشير بشكلٍ متكرر إلى "[sdb]" التي تطابق توقعاتنا بإظهار اسم جهاز SCSI. السطرين التاليين مثيرين للاهتمام بشكل خاص:

```
Jul 23 10:07:59 linuxbox kernel: sdb: sdb1
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Attached SCSI
removable disk
```

هذا يخبرنا أن اسم الجهاز هو `/dev/sdb` لكامل الجهاز و`/dev/sdb1` للقطاع الأول من الجهاز.

ملاحظة: الأمر `tail -f /var/log/messages` رائع لمراقبة ما يحدث في النظام بالوقت الحقيقي.

وبعد أن عرفنا اسم الجهاز، حان الآن وقت وصله:

```
[me@linuxbox ~]$ sudo mkdir /mnt/flash
[me@linuxbox ~]$ sudo mount /dev/sdb1 /mnt/flash
[me@linuxbox ~]$ df
Filesystem       1K-blocks      Used    Available  Use%  Mounted on
/dev/sda2         15115452   5186944      9775164  35%   /
/dev/sda5         59631908  31777376     24776480  57%   /home
/dev/sda1           147764     17277       122858  13%   /boot
tmpfs               776808         0       776808   0%   /dev/shm
/dev/sdb1            15560         0        15560   0%   /mnt/flash
```

سيبقى اسم الجهاز نفسه لطالما أن الجهاز بقي متصلًا بالحاسوب ولم يُعَد إقلاع النظام.

إنشاء أنظمة ملفات جديدة

لنفترض أننا نريد تهيئة قرص الفلاش بنظام ملفات لينُكس الأصلي (يقصد به هنا ext3) بدلًا من نظام ملفات FAT32 الحالي. هذا يتضمن خطوتَين: 1- (اختيارية) إنشاء جدول قطاعات جديد إذا لم يعجبنا الجدول الحالي. و 2- إنشاء نظام ملفات فارغ وجديد على القرص.

تحذير! ستُهيئ، في التمرين الآتي، قرص فلاش. استخدم قرصًا لا يحتوي على أي شيء مهم بالنسبة لك لأنه سيُمحى! أعيد وأكرر، **تحقق بنسبة 100% من أنك تحدد اسم الجهاز الصحيح وليس الاسم الموجود في الأمثلة. سيؤدي الإخفاق في الالتزام بهذا التحذير إلى تهيئة القرص الخطأ!**

تعديل الأقسام باستخدام fdisk

يسمح البرنامج fdisk لنا بالتعامل مع الأقراص التخزينية (كالقرص الصلب والفلاش) تعاملًا منخفض المستوى. نستطيع، باستخدام هذه الأداة، أن نعدل ونحذف وننشئ القطاعات على الجهاز. لبدء العمل مع قرص الفلاش، سنفصله أولًا (إذا لزم الأمر) ومن ثم نُشغِّل برنامج fdisk كالآتي:

```
[me@linuxbox ~]$ sudo umount /dev/sdb1
[me@linuxbox ~]$ sudo fdisk /dev/sdb
```

لاحظ أننا حددنا الجهاز بأكمله وليس أحد أقسامه. سنشاهد المِحث الآتي بعد بدء البرنامج:

```
Command (m for help):
```

سيؤدي الضغط على "m" إلى إظهار قائمة البرنامج:

```
Command action
    a    toggle a bootable flag
    b    edit bsd disklabel
    c    toggle the dos compatibility flag
    d    delete a partition
    l    list known partition types
    m    print this menu
    n    add a new partition
    o    create a new empty DOS partition table
    p    print the partition table
    q    quit without saving changes
    s    create a new empty Sun disklabel
    t    change a partition's system id
    u    change display/entry units
    v    verify the partition table
    w    write table to disk and exit
    x    extra functionality (experts only)



Command (m for help):
```

أول ما سنفعله هو مشاهدة جدول القطاعات الحالي. نستطيع فعل ذلك بالضغط على حرف "p":

```
Command (m for help): p
Disk /dev/sdb: 16 MB, 16006656 bytes
1 heads, 31 sectors/track, 1008 cylinders
Units = cylinders of 31 * 512 = 15872 bytes

   Device Boot    Start      End     Blocks    Id   System
/dev/sdb1             2      1008    15608+     b   W95 FAT32
```

نشاهد في المثال السابق جهازًا بسعة تخزينية 16 MB بقطاع وحيد (1) الذي يستخدم 1006 من أصل 1008 أسطوانة (cylinders) متوفرة على الجهاز. تم التعرف على القطاع على أنه قطاع "Windows 95 FAT32". تستخدم بعض البرامج تلك المعلومة لكي يجعلوا العمليات التي يمكن تنفيذها على القرص محددة ومحدودة. في أغلب الوقت، لن يكون هنالك طائل من تغييرها، لكن ولغرض هذا الفصل، سنغيرها إلى قطاع بنظام ملفات

ext3. للقيام بذلك، يجب علينا أن نعرف ما هو ID المُستخدَم لتعريف قطاع لينُكس. في المثال السابق نجد أن ID هو "b". لعرض قائمة بكامل أنواع القطاعات المتوفرة، فإننا نعود إلى القائمة الرئيسية. ونشاهد الخيار الآتي:

```
l  list known partition types
```

إذا أدخلنا "l" في المِحث، فستُطبَع مجموعة كبيرة من الأنواع المتوفرة. عند البحث فيهم نجد "b" الذي يشير إلى نوع القطاع الحالي و "83" لنظام ملفات لينُكس.

نستطيع مشاهدة هذا الخيار لتغيير ID الخاص بقطاعٍ ما بعد العودة إلى القائمة الرئيسية:

```
t  change a partition's system id
```

بعد كتابة "t" في المِحث، فإننا نُدخِل القيمة الجديدة لمُعَرِّف نظام الملفات:

```
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): 83
Changed system type of partition 1 to 83 (Linux)
```

لقد انتهينا الآن من جميع التعديلات التي نريدها. لهذا الحد، لم يلمس الجهاز أبدًا (جميع التغييرات خُزِّنَت في الذاكرة وليس على الجهاز الفيزيائي)، لكتابة جدول القطاعات المعدل وإنهاء البرنامج، اطبع "w" في المِحث:

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
[me@linuxbox ~]$
```

إذا قررنا عدم تعديل الجهاز فإننا نُدخِل "q" في المِحث، وهذا ما يقوم بإنهاء البرنامج دون كتابة التعديلات. يمكننا تجاهل رسالة الخطأ السابقة.

إنشاء نظام ملفات جديد باستخدام mkfs

وبعد أن انتهينا من تعديل القطاعات، حان الوقت لإنشاء نظام ملفات جديد في قرص الفلاش الخاص بنا. للقيام بذلك، سنستخدم الأمر mkfs (اختصار للعبارة "make file system") الذي يمكنه إنشاء العديد من أنواع أنظمة الملفات. لإنشاء نظام ملفات ext3 على الجهاز فإننا نستخدم الخيار "t-" لتحديد نوع نظام الملفات "ext3" ويلحقه اسم الجهاز الذي يحتوي على القسم الذي نريد تهيئته:

```
[me@linuxbox ~]$ sudo mkfs -t ext3 /dev/sdb1
mke2fs 1.40.2 (12-Jul-2007)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
3904 inodes, 15608 blocks
780 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=15990784
2 block groups
8192 blocks per group, 8192 fragments per group
1952 inodes per group
Superblock backups stored on blocks:
     8193

Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 34 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
[me@linuxbox ~]$
```

سيعرض البرنامج العديد من المعلومات عند اختيار "ext3" كنوع نظام الملفات. لإعادة تهيئة الجهاز بنظام ملفات FAT32 الأصلي فإننا نحدد "vfat" كنوع نظام الملفات:

```
[me@linuxbox ~]$ sudo mkfs -t vfat /dev/sdb1
```

يمكن استخدام عملية التقطيع والتهيئة مع أي جهاز تخزيني يضاف للحاسوب. بينما جربنا على قرص فلاش

صغير، إلا أن العملية هي ذاتها عند تطبيقها على الأقراص الصلبة وباقي وسائط التخزين.

تفحص وإصلاح أنظمة الملفات

رأينا أرقامًا غامضةً في آخر كل سطر في نقاشنا السابق حول ملف /etc/fstab. يتفحص النظام الأقراص بشكل روتيني قبل وصلها في كل مرّة يُقلع فيها. يتم ذلك ببرنامج fsck (اختصار للعبارة "file system check"). يُحَدِّد الرقم الأخير الموجود في كل سطر في ملف fstab الترتيب الذي سيتبعه النظام لتفحص ذاك القرص. في ذاك المثال، يمكننا ملاحظة أن نظام الملفات الجذر هو أول ما يتم تفحصه، ويتبعه أنظمة ملفات home و boot. لا يتم تفحص الأقراص التي ينتهي سطرها بالرقم "0" (صفر).

بالإضافة إلى تفحص الأقراص، يستطيع fsck أيضًا إصلاح أنظمة الملفات المعطوبة بدرجاتٍ متفاوتةً من النجاح، وذلك وفقًا لمقدار الضرر الحاصل على القطاع. توضع الملفات المستعادة في الأنظمة الشبيهة بِيونكس في مجلد lost+found الموجود في المجلد الجذر لنظام الملفات.

لتفحص قرص الفلاش الخاص بنا (الذي يجب فصله أولًا):

```
[me@linuxbox ~]$ sudo fsck /dev/sdb1
fsck 1.40.8 (13-Mar-2008)
e2fsck 1.40.8 (13-Mar-2008)
/dev/sdb1: clean, 11/3904 files, 1661/15608 blocks
```

حسب خبرتي، إن حدوث ضرر في نظام الملفات هو حالة نادرة إلا في حال وجود مشكلة تتعلق بالعتاد. يُعرَف في أغلب الأنظمة أن نظام الملفات متضرر في وقت الإقلاع حيث يؤدي ذلك إلى توقف الإقلاع وسيوجهك النظام إلى تشغيل fsck قبل الإكمال.

تهيئة الأقراص المرنة

ما يزال العديد منّا يستخدم حواسيب قديمة ما تزال تحتوي على قارئ أقراص مرنة، يمكننا إدارة تلك الأقراص أيضًا. تحضير قرص مرن للاستخدام هو عملية تتألف من خطوتَين: أولًا سنقوم بتهيئة منخفضة المستوى على القرص، ومن ثم سننشئ نظام الملفات. نستخدم برنامج fdformat للقيام بعملية التهيئة محددين اسم القرص المرن (يكون عادةً /dev/fd0):

```
[me@linuxbox ~]$ sudo fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
```

لنُنشئ الآن نظام ملفات FAT في القرص المرن باستخدام الأمر `mkfs`:

```
[me@linuxbox ~]$ sudo mkfs -t msdos /dev/fd0
```

لاحظ أننا استخدمنا نظام الملفات "msdos" للحصول على نمط جدول الملفات القديم (والصغير). يكون القرص المرن جاهزًا للوصل والاستخدام كباقي الأجهزة بعد أن تُنفَّذ الأوامر السابقة.

نقل البيانات مباشرةً من وإلى الأجهزة

على الرغم من أننا نتخيل البيانات المخزنة على حواسيبنا منظمة على شكل ملفات، إلا أننا نستطيع أيضًا أن نتخيلها وهي بالشكل "الخام" (raw). إذا نظرنا إلى محرك الأقراص، على سبيل المثال، فإننا سنجده مليئًا "بكتل" (blocks) البيانات التي يراها نظام التشغيل مجلدات وملفات. لكننا إذا عاملنا محرك الأقراص على أنه مجموعة ضخمة من كتل البيانات فإننا نستطيع أن نجري مهمات مفيدة كنسخ الأقراص.

البرنامج dd يقوم بتلك المهمة. حيث ينسخ كتل من البيانات من مكان إلى آخر. ويتميز بصياغة خاصة به (وذلك لأسباب تاريخية) ويُستخدم عادةً بهذه الطريقة:

```
dd if=input_file of=output_file [bs=block_size [count=blocks]]
```

لنفترض أن لدينا قرصَيّ USB بنفس الحجم التخزيني ونريد إنشاء نسخة طبق الأصل من أول قرص إلى الآخر. إذا أضفنا القرصَين إلى الحاسوب فإنهما سيأخذان اسمي الجهازَين `/dev/sdb` و `/dev/sdc` على التوالي، نستطيع نسخ كل شيء من القرص الأول إلى الثاني باستخدام الأمر الآتي:

```
dd if=/dev/sdb of=/dev/sdc
```

بشكلٍ بديل، إذا كان القرص الأول موصولًا إلى الحاسوب فقط، فيمكننا أن ننسخ محتواه إلى ملفٍ عادي للاسترجاع أو النسخ لاحقًا:

```
dd if=/dev/sdb of=flash_driver.img
```

تحذير! الأمر dd قوي جدًا. وعلى الرغم من أن اسمه هو اختصار لعبارة "data definition" إلا أنه يسمى عادةً "destroy disk" لأن المستخدمين يخطئون عادةً في تعبير `if` أو `of`. **دائمًا تأكد من أسماء الأجهزة بدقة قبل الضغط على Enter!**

إنشاء صور أقراص CD-ROM

تكون آلية كتابة قرص CD-ROM (إما أن يكون CD-R أو CD-RW) متألفة من خطوتَين: الأولى هي إنشاء صورة قرص iso (image) الذي هو عبارة عن صورة نظام الملفات الذي سيكتب على قرص CD والخطوة الثانية هي كتابة ملف الصورة إلى قرص CD.

إنشاء صورة قرص من CD-ROM

إذا أردت إنشاء ملف صورة قرص من قرص CD-ROM موجود مسبقًا، فيمكنك استخدام الأمر dd لقراءة جميع كتل البيانات في القرص ونسخها إلى ملفٍ محلي. لنفترض أن لدينا قرص CD لتوزيعة أوبنتو ونريد إنشاء ملف iso كي نستطيع إنشاء عدّة نسخ لاحقًا. بعد إدراج القرص وتحديد اسم الجهاز (سنفترض أنه/dev/cdrom)، فإننا ننشئ الملف على النحو الآتي:

```
dd if=/dev/cdrom of=ubuntu.iso
```

هذه الطريقة تعمل مع أقراص DVD أيضًا، لكنها لن تعمل مع اقرص "audio CD"؛ لأنها لا تستخدم نظام ملفات للتخزين. نستخدم الأمر cdrdao لأقراص audio CD.

إنشاء صورة قرص من مجموعة من الملفات

سنستخدم البرنامج grenisoimage. لإنشاء صورة قرص تحتوي على محتويات مجلدٍ ما. ننشئ أولًا مجلدًا يحتوي على جميع الملفات التي نريد تضمينها في صورة القرص ومن ثم ننفذ الأمر genisoimage لإنشاء ملف الصورة. على سبيل المثال، ليكن لدينا مجلد يسمى cd-rom-files/~ يحتوي على الملفات التي نريد كتابتها إلى قرص CD-ROM، فبإمكاننا إنشاء ملف صورة يدعى cd-rom.iso بالأمر الآتي:

```
genisoimage -o cd-rom.iso -R -J ~/cd-rom-files
```

يضيف الخيار "R-" البيانات الوصفية بالإضافة إلى السماح باستخدام أسماء الملفات الطويلة وأذونات الملفات من نمط POSIX. وبشكلٍ مشابه، يُفعِّل الخيار "J-" الدعم لأسماء الملفات الطويلة لنظام ويندوز.

نفس البرنامج لكن باسم آخر...

إذا ألقيتَ نظرة على الدروس التعليمية في الإنترنت لإنشاء وحرق الأقراص الضوئية كأقراص CD-ROM و DVD، سيمر عليك اِسمَيّ برنامجَين هما "mkisofs" و "cdrecord". هذان البرنامجان هما جزء من حزمة باسم "cdrtools" طُوِرَت من قِبل Jorg Schilling. في صيف عام 2006، غَيّر

Schilling رخصة تلك الحزمة، الأمر الذي اعتبره الكثيرون من مجتمع لينُكس على أنه يجعل رخصة cdrtools غير متوافقة مع رخصة غنو العمومية GNU GPL. نتيجةً لذلك، أُشتقّ (fork) مشروع cdrtools، وأصبحت أسماء البرامج البديلة هي wodim و genisoimage بدلًا من cdrecord و mkisofs على التوالي.

كتابة صور أقراص CD-ROM

بعد أن أنشأنا ملف الصورة، يمكننا الآن حرقه إلى القرص الضوئي. أغلب الأوامر التي سنناقشها تنطبق على أقراص CD-ROM و DVD القابلة للكتابة.

وصل ملف صورة قرص ISO مباشرةً

هنالك خدعة صغيرة تسمح لنا بوصل ملف iso بينما هو موجود في قرصنا الصلب ويعامل من قِبل النظام كقرص ضوئي. يتم ذلك بالخيار "-o loop" في الأمر mount (بالإضافة إلى تحديد نوع نظام الملفات "-t iso9660"):

```
mkdir /mnt/iso_image
mount -t iso9660 -o loop image.iso /mnt/iso_image
```

أنشأنا في المثال السابق، نقطة الوصل /mnt/iso_image ومن ثم وصلنا صورة القرص ذات الاسم image.iso إلى تلك النقطة. بعد وصل صورة القرص، ستُعامل كأنها قرص CD-ROM أو DVD حقيقي. تذكر أن تفصل الصورة عند عدم الحاجة لها.

محو البيانات على قرص قابل لإعادة الكتابة

تحتاج أقراص CD-RW القابلة لإعادة الكتابة إلى محو محتوياتها قبل إعادة الكتابة عليها. نستخدم البرنامج wodim للقيام بذلك؛ محددين اسم الجهاز الذي يحتوي على قرص CD-RW ونوع المحو الذي سيتم. يوفر برنامج wodim العديد من الأنواع. أسرع تلك الأنواع هو "fast":

```
wodim dev=/dev/cdrw blank=fast
```

كتابة صورة القرص

نستخدم البرنامج wodim مرّة أخرى لكتابة ملف صورة القرص، لكن هذه المرّة مع تحديد مسار صورة القرص كوسيط كالآتي:

```
wodim dev=/dev/cdrw image.iso
```

بالإضافة إلى اسم الجهاز ومسار صورة القرص، يدعم برنامج wodim تشكيلةً كبيرةً من الخيارات. أشهر تلك الخيارات هي الخيار "v-" لإظهار مخرجات عن نسبة التقدم، والخيار "dao-" الذي يكتب القرص بنمط disk-at-once. يجب استخدام هذا النمط في حال أردت استخدام القرص الناتج في أعمال تجارية. النمط الافتراضي لبرنامج wodim هو track-at-once الذي يكون مفيدًا عند نسخ مقاطع موسيقية على سبيل المثال.

الخلاصة

لقد ألقينا نظرة في هذا الفصل على المهام الأساسية لإدارة الأقراص. ما زال هنالك الكثير. يدعم لينُكس عددًا كبيرًا جدًا من أجهزة التخزين وأنظمة الملفات. ويوفر أيضًا العديد من الميزات للتوافق مع الأنظمة الأُخرى.

أضف إلى معلوماتك

من المفيد عادةً أن نتحقق من سلامة محتويات ملف iso نُزّل من الإنترنت. في أغلب الحالات يُوفِّر موزع القرص "ملف بصمة". البصمة هي ناتج عملية رياضية معقدة تُنتج رقمًا متعلقًا بمحتوى الملف، إذا تغير محتوى الملف ولو بتًا واحدًا، فإن البصمة ستختلف. أشهر طريقة لتوليد البصمات هي استخدام البرنامج md5sum. عندما تستخدم md5sum فسيظهر لك عدد ست عشري:

```
md5sum image.iso
34e354760f9bb7fbf85c96f6a3f94ece  image.iso
```

بعد أن تنتهي من تنزيل ملف iso من الإنترنت، يجب عليك أن تقارن ناتج الأمر md5sum برقم البصمة التي وفرها الموزع.

بالإضافة إلى التحقق من سلامة التنزيل، يمكننا استخدام md5sum للتحقق من سلامة البيانات التي كُتِبَت حديثًا إلى قرص ضوئي. للقيام بذلك، يجب أن نحسب أولًا بصمة الصورة، ومن ثم نحسب البصمة للقرص الضوئي ومن ثم نقارنهما سويةً. الخدعة التي سنستخدمها للتحقق من البيانات هي حساب البصمة لجزء من القرص الذي يحوي على الصورة. للقيام بذلك نقوم باعتماد 2048 بايت من القرص (الأقراص الضوئية دائمًا تُكتب بشكل كتل 2048 بايت). حساب البصمة لا يتطلب ذلك في بعض الأقراص كقرص CD-R كُتِبَ عليه بنمط disk-at-once:

```
md5sum /dev/cdrom
34e354760f9bb7fbf85c96f6a3f94ece  /dev/cdrom
```

العديد من الأقراص الأخرى تحتاج إلى بعض الحسابات لتحديد عدد الكتل. في المثال الآتي، سنتحقق من سلامة قرص DVD موجود في dev/dvd/. هل تستطيع معرفة كيف تم ذلك؟

```
md5sum dvd-image.iso; dd if=/dev/dvd bs=2048 count=$(( $(stat -c "%s"
dvd-image.iso) / 2048 )) | md5sum
```

الفصل السادس عشر:

الشبكات

عندما يأتي الأمر للشبكات، فلن تجد أيّة مهمة لا يمكن القيام بها في لينُكس. يُستخدم لينُكس لبناء جميع أنظمة وأجهزة الشبكات، بما فيها الجدران النارية والموجهات (routers) وخوادم الأسماء و NAS ("Network Attached Storage" التي تعني التخزين عبر الشبكة) ...إلخ.

ولأن موضوع الشبكات هو موضوع واسع، وكذلك الأوامر التي تُستخدم لإعداد الشبكة والتحكم فيها؛ سنركز نقاشنا على بعض الأوامر الشائعة الاستخدام. الأدوات التي سنشرحها تتضمن مراقبة الشبكات ونقل الملفات. إضافةً إلى ذلك، سنستكشف برنامج `ssh` الذي يُستخدم للقيام بعمليات تسجيل دخول بعيدة. سيشرح هذا الفصل:

- `ping` – إرسال ICMP ECHO_REQUEST إلى أجهزة الشبكة.
- `traceroute` – طباعة الطريق الذي تسلكه حزم البيانات كي تصل إلى الجهاز الهدف.
- `netstat` – طباعة اتصالات الشبكة، وجداول التوجيه (routing tables)، وإحصائيات عن أجهزة الاتصال المستخدمة للوصول إلى الشبكة ...إلخ.
- `ftp` – برنامج نقل الملفات عبر الإنترنت.
- `wget` – مُنَزِّل شبكي غير تفاعلي.
- `ssh` – عميل SSH (يُستخدم للدخول إلى النظام عن بعد).

سنفترض أن لديك خلفية قليلة في الشبكات. في عصر الإنترنت، سيحتاج كل شخص يستخدم الحاسوب إلى قليل من المعرفة حول أساسيات الشبكة. يجب أن تكون المصطلحات الآتية مألوفةً لديك للاستفادة لأقصى حد من هذا الفصل:

- عنوان `IP` (Internet Protocol).
- المضيف (host) واسم النطاق (domain name).
- URI (Uniform Resource Identifier).

ملاحظة: قد تحتاج بعض الأوامر التي سنناقشها إلى تثبيت (حسب توزيعتك) بعض الحزم الإضافية من المستودعات، ويحتاج بعضها الآخر إلى امتيازات الجذر لكي تُنفَّذ.

استكشاف ومراقبة الشبكة

حتى وإن لم تكن مديرًا للنظام، ستستفيد من مراقبة أداء وسير العمليات في الشبكة.

ping

أبسط أمر يتعلق بالشبكة هو ping. يُرسِل الأمر ping حزمةً شبكيةً (network packet) تدعى IMCP ECHO_REQUEST لجهاز معين. تردّ أغلب أجهزة الشبكة عند تلقيها لهذه الحزمة، يؤدي ذلك إلى التحقق من اتصال الشبكة.

ملاحظة: من الممكن أن تُضبَط أغلب أجهزة الشبكة (بما فيها الأجهزة التي تعمل على نظام تشغيل لينُكس) لكي تتجاهل هذه الحِزم. يُفعَل ذلك عادةً لأسباب أمنية، إذا أردنا منع المهاجم من رؤية جهازٍ ما، فعلينا ضبط إعدادات الجدار الناري لكي يمنع اتصالات IMCP.

على سبيل المثال، إن أردنا معرفة إذا كنا نستطيع الوصول إلى الخادم linuxcommand.org، فسنستخدم الأمر ping كالآتي:

```
[me@linuxbox ~]$ ping linuxcommand.org
```

سيستمر ping في إرسال الحِزم (بعد تشغيله) بعد مرور فاصل زمني معين (الفاصل الافتراضي هو ثانية واحدة) إلى أن يتم إنهاؤه:

```
[me@linuxbox ~]$ ping linuxcommand.org
PING linuxcommand.org (66.35.250.210) 56(84) bytes of data.
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=1
ttl=43 time=107 ms
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=2
ttl=43 time=108 ms
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=3
ttl=43 time=106 ms
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=4
ttl=43 time=106 ms
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=5
ttl=43 time=105 ms
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=6
ttl=43 time=107 ms
```

```
--- linuxcommand.org ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 6010ms
rtt min/avg/max/mdev = 105.647/107.052/108.118/0.824 ms
```

بعد أن يتم إنهاء الأمر بالضغط على `Ctrl-c` (في هذه الحالة، بعد إرسال الحزمة السادسة)، يطبع `ping` إحصائيات عن الأداء. يجب أن تكون نسبة فقدان الحزم في شبكة ذات أداء مقبول هي 0%. عملية "ping" ناجحة تعني أن جميع مكونات الشبكة (أي بطاقات الشبكة والكابلات والموجهات والبوابات) تعمل جيدًا.

traceroute

يُظهِر برنامج `traceroute` (بعض الأنظمة تستخدم البرنامج `tracepath` بدلًا منه) قائمة بجميع "العقد" (hobs) التي تمر منها الحِزم الشبكية حتى تصل إلى الجهاز الهدف. على سبيل المثال، لعرض الطريق الذي يُسلَك للوصول إلى موقع `slashdot.org`:

```
[me@linuxbox ~]$ traceroute slashdot.org
```

ستظهر مخرجات شبيهة بالآتي:

```
traceroute to slashdot.org (216.34.181.45), 30 hops max, 40 byte
packets
 1 ipcop.localdomain (192.168.1.1) 1.066 ms 1.366 ms 1.720 ms
 2 * * *
 3 ge-4-13-ur01.rockville.md.bad.comcast.net (68.87.130.9) 14.622
ms 14.885 ms 15.169 ms
 4 po-30-ur02.rockville.md.bad.comcast.net (68.87.129.154) 17.634
ms 17.626 ms 17.899 ms
 5 po-60-ur03.rockville.md.bad.comcast.net (68.87.129.158) 15.992
ms 15.983 ms 16.256 ms
 6 po-30-ar01.howardcounty.md.bad.comcast.net (68.87.136.5) 22.835
ms 14.233 ms 14.405 ms
 7 po-10-ar02.whitemarsh.md.bad.comcast.net (68.87.129.34) 16.154
ms 13.600 ms 18.867 ms
 8 te-0-3-0-1-cr01.philadelphia.pa.ibone.comcast.net (68.86.90.77)
21.951 ms 21.073 ms 21.557 ms
 9 pos-0-8-0-0-cr01.newyork.ny.ibone.comcast.net (68.86.85.10)
22.917 ms 21.884 ms 22.126 ms
```

```
 10 204.70.144.1 (204.70.144.1) 43.110 ms 21.248 ms 21.264 ms
 11 cr1-pos-0-7-3-1.newyork.savvis.net (204.70.195.93) 21.857 ms
cr2-pos-0-0-3-1.newyork.savvis.net (204.70.204.238) 19.556 ms cr1-
pos-0-7-3-1.newyork.savvis.net (204.70.195.93) 19.634 ms
 12 cr2-pos-0-7-3-0.chicago.savvis.net (204.70.192.109) 41.586 ms
 42.843 ms cr2-tengig-0-0-2-0.chicago.savvis.net (204.70.196.242)
 43.115 ms
 13 hr2-tengigabitethernet-12-1.elkgrovech3.savvis.net
(204.70.195.122) 44.215 ms 41.833 ms 45.658 ms
 14 csr1-ve241.elkgrovech3.savvis.net (216.64.194.42) 46.840 ms
 43.372 ms 47.041 ms
 15 64.27.160.194 (64.27.160.194) 56.137 ms 55.887 ms 52.810 ms
 16 slashdot.org (216.34.181.45) 42.727 ms 42.016 ms 41.437 ms
```

يمكننا ملاحظة، من المخرجات السابقة، أن اتصال نظامنا بالمضيف slashdot.org يتطلب المرور بستة موجهات. نستطيع مشاهدة اسم المضيف في الموجهات التي توفر معلومات عنها بالإضافة إلى عنوان ip ومعلومات حول الأداء التي تحتوي على الوقت المستغرق للحزم لتنتقل من جهازنا المحلي إلى ذاك الموجه. بالنسبة إلى الموجهات التي لا توفر معلومات عنها (بسبب إعدادات الموجه أو الجدران النارية ...إلخ.)، سنشاهد رمز النجمة "*" في السطر الذي يوافق ذاك الموجه (السطر الثاني في المثال السابق).

netstat

يُستخدم برنامج netstat للحصول على معلوماتٍ مختلفة حول إعدادات الشبكة وبعض الإحصائيات. نستطيع، باستخدام العدد الكبير من الخيارات الخاصة بالبرنامج، مشاهدة العديد من المعلومات حول ضبط الشبكة. ونستطيع أيضًا أن نستكشف بطاقات الشبكة (أو أيّة طريقة اتصال أُخرى) المتوفرة في جهازنا باستخدام الخيار "ie-":

```
[me@linuxbox ~]$ netstat -ie
eth0      Link encap:Ethernet HWaddr 00:1d:09:9b:99:67
          inet addr:192.168.1.2 Bcast:192.168.1.255 Mask:255.255.255.0
          inet6 addr: fe80::21d:9ff:fe9b:9967/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:238488 errors:0 dropped:0 overruns:0 frame:0
          TX packets:403217 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
```

```
          RX bytes:153098921 (146.0 MB) TX bytes:261035246 (248.9 MB)
          Memory:fdfc0000-fdfe0000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:2208 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2208 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:111490 (108.8 KB) TX bytes:111490 (108.8 KB)
```

لاحظنا في المثال السابق أن النظام الذي نجرّب عليه يحتوي على بطاقتَيّ شبكة. الأولى التي تسمى eth0 هي منفذ كبل الشبكة من نوع Ethernet. أما الثانية فتسمى lo والتي تعني loopback interface، وهي عبارة عن منفذ افتراضي يسمح للنظام بأن "يتكلم مع نفسه".

عندما نقوم بعملية "تشخيص" اعتيادية للشبكة، فإن الأشياء المهمة التي يجب البحث عنها هي وجود كلمة "UP" في بداية السطر الرابع من كل بطاقة شبكة، التي تعني أن بطاقة الشبكة مفعلة. وأيضًا وجود عنوان ip صحيح في حقل inet addr في السطر الثاني. للأنظمة التي تستخدم DHCP (Dunamic Host Configuration Protocol)، وجود عنوان ip صحيح يعني أن ميزة DHCP تعمل دون مشاكل.

سيعرض الخيار "r-" عند استخدامه؛ جدول التوجيهات الخاص بالنواة. الذي يعرض كيف أُعِدّت الشبكة لإرسال الحزم من شبكة لأخرى:

```
[me@linuxbox ~]$ netstat -r
Kernel IP routing table
Destination    Gateway     Genmask       Flags  MSS  Window irtt  Iface
192.168.1.0    *           255.255.255.0 U         0 0         0  eth0
default        192.168.1.1 0.0.0.0       UG        0 0         0  eth0
```

شاهدنا، في المثال السابق، جدول توجيهات اعتيادي لجهاز العميل في شبكة محلية LAN (Local Area Network) خلف جدار ناري/موجه. أول سطر من الناتج السابق يظهر أن الهدف هو 192.168.1.0. عناوين ip التي تنتهي بصفر تُشير إلى الشبكات وليس إلى الأجهزة نفسها، لذا، فإن الوجهة هي أي جهاز في الشبكة المحلية LAN. الحقل التالي "Gateway" هو اسم أو عنوان ip للبوابة (الموجه router) التي تُستخدم للذهاب من الجهاز المحلي الحالي إلى الشبكة الهدف. وجود نجمة "*" في هذا الحقل تعني أنه لا حاجة إلى بوابة.

السطر الأخير يحدد الوجهة default. أي وجهة أيّة بيانات تُرسَل إلى شبكة لم تذكر في هذا الجدول. ويمكننا

ملاحظة أن البوابة قد عُرِّفَت على أنها موجه ذو العنوان 192.168.1.1 الذي قد يعرف ما الذي عليه فعله مع تلك البيانات.

لدى برنامج `netstat` العديد من الخيارات ولم نجرب سوى اثنين منهما. تفقد صفحة الدليل لبرنامج `netstat` للحصول على القائمة الكاملة.

نقل الملفات عبر الشبكة

ما هي الفائدة من الشبكات إذا لم نكن نعرف كيف ننقل الملفات عبرها؟ يوجد العديد من البرامج التي تنقل البيانات عبر الشبكة. سنشرح إثنين منهما الآن وسنشرح الباقي لاحقًا.

ftp

أحد أكثر البرامج "الكلاسيكية" هو `ftp`، يأتي اسم `ftp` من اسم البروتوكول الذي يستخدمه، File Transfer Protocol، يُستخدم FTP كثيرًا لتنزيل الملفات من الإنترنت. تدعمه أغلب، إن لم يكن جميع، متصفحات الوِب. ربما شاهدتَ العديد من الروابط التي تشير إلى `ftp://`.

قبل وجود متصفحات الوِب، كان هنالك برنامج `ftp`. يُستخدم `ftp` للتواصل مع خوادم FTP، تلك الأجهزة التي تحتوي على ملفات يمكن رفعها أو تنزيلها عبر الشبكة.

الاستخدام التقليدي لبروتوكول FTP غير آمن لأن أسماء المستخدمين وكلمات مرورهم تُرسَل عبر الشبكة كما هي دون أي تشفير، يستطيع أي شخص "يتنصت" على الشبكة مشاهدتهم. ولهذا السبب، تقبل جميع خوادم FTP تقريبًا استخدام الهوية المجهولة (anonymous FTP)، مما يسمح لأي شخص بالدخول بحساب "anonymous" وبأية كلمة مرور يختارها.

في المثال الآتي، سنستعرض جلسة اعتيادية في برنامج `ftp` لتنزيل صورة `iso` لقرص أوبنتو الموجودة في مجلد `/pub/cd_images/Ubuntu-8.04` في خادم FTP الذي عنوانه هو `fileserver`:

```
[me@linuxbox ~]$ ftp fileserver
Connected to fileserver.localdomain.
220 (vsFTPd 2.0.1)
Name (fileserver:me): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
```

```
ftp> cd pub/cd_images/Ubuntu-8.04
250 Directory successfully changed.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-rw-r--  1 500  500  733079552 Apr 25 03:53 ubuntu-8.04-desktop-
i386.iso
226 Directory send OK.
ftp> lcd Desktop
Local directory now /home/me/Desktop
ftp> get ubuntu-8.04-desktop-i386.iso
local: ubuntu-8.04-desktop-i386.iso remote: ubuntu-8.04-desktop-
i386.iso
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for ubuntu-8.04-desktop-
i386.iso (733079552 bytes).
226 File send OK.
733079552 bytes received in 68.56 secs (10441.5 kB/s)
ftp> bye
```

يشرح الجدول الآتي الأوامر التي أُدخلت خلال الجلسة:

الجدول 16-1: الأوامر التي أُدخلت أثناء جلسة ftp

الأمر	الشرح
`ftp fileserver`	استخدام البرنامج ftp للاتصال بخادم fileserver.
`anonymous`	اسم تسجيل الدخول. بعد مِحث الدخول، سيظهر مِحث يطلب كلمة المرور. بعض الخوادم تقبل كلمة مرور فارغة، البعض الآخر يتطلب أن تكون كلمة المرور على شكل عنوان بريد إلكتروني "user@example.com".
`cd pub/cd_images/Ubuntu-8.04`	تغيير المجلد في النظام البعيد إلى المجلد الذي يحتوي على الملف المطلوب. لاحظ أن الملفات المتاحة للتنزيل من قِبل الجميع تُوضَع في أغلب الخوادم التي تسمح بالدخول بهوية مجهولة (anonymous) في مكانٍ ما في مجلد pub.

ls	عرض محتويات المجلد في النظام البعيد.
lcd Desktop	تبديل مجلد العمل في النظام المحلي إلى ‎~/Desktop‎. شُغِّلَ برنامج ftp عندما كان مجلد العمل الحالي هو "~". هذا الأمر سيغيره إلى ‎~/Desktop‎.
get ubuntu-8.04-desktop-i386.iso	إخبار النظام البعيد أننا نريد نقل الملف ubuntu-8.04-desktop-i386.iso إلى نظامنا المحلي. ولأن مجلد العمل الحالي في النظام المحلي هو ‎~/Desktop‎، فسينزّل الملف هناك.
bye	تسجيل الخروج من الخادم وإنهاء جلسة برنامج ftp. يمكن استخدام الأمرين quit و exit أيضًا.

تؤدي طباعة "help" في مِحث "ftp>" إلى إظهار قائمة بالأوامر المدعومة. يمكنك استخدام ftp على خادم تمتلك امتيازات كافية فيه للقيام بالعديد من أعمال إدارة الملفات.

lftp: عميل ftp مُحسّن

برنامج ftp ليس عميل ftp الوحيد الذي يعمل من سطر الأوامر. يوجد بديل أفضل (وأشهر أيضًا) هو lftp. يعمل عملًا مشابهًا لعميل ftp التقليدي، إلا أنه يدعم العديد من الميزات الأخرى كدعم عدة بروتوكولات، وإعادة المحاولة عند فشل التنزيل، واستخدام العمليات في الخلفية، والإكمال التلقائي لأسماء الملفات (باستخدام الزر tab) والكثير.

wget

برنامج سطر أوامر آخر شهير لتنزيل الملفات هو wget. وهو أداة مفيدة جدًا عند تنزيل ملف واحد أو ملفات متعددة من مواقع وِب أو FTP. وحتى مواقع بأكملها! لتنزيل الصفحة الرئيسية لموقع linuxcommand.org ننفذ الأمر الآتي:

```
[me@linuxbox ~]$ wget http://linuxcommand.org/index.php
--11:02:51-- http://linuxcommand.org/index.php
           => `index.php'
Resolving linuxcommand.org... 66.35.250.210
Connecting to linuxcommand.org|66.35.250.210|:80... connected.
```

```
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
     [    <=>                                ] 3,120         --.--K/s
11:02:51 (161.75 MB/s) - `index.php' saved [3120]
```

تسمح الخيارات الكثيرة لبرنامج `wget` بتنزيل المواقع بأكملها أو تنزيل الملفات في الخلفية (أي أنه يسمح لك بتسجيل الخروج مع الإبقاء على عملية التنزيل قائمةً) وإكمال تنزيل ملفٍ نُزلَّ تنزيلًا جُزئيًّا. كل هذه الخيارات موثقة جيدًا في صفحة الدليل الخاصة به.

الاتصالات الآمنة مع الأجهزة البعيدة

منذ العديد من السنوات، كانت الأنظمة الشبيهة بِيونكس قابلة للإدارة عن بعد باستخدام الشبكة. في تلك الأيام وقبل انتشار الإنترنت انتشارًا كبيرًا؛ كان هنالك برنامجان أساسيان لتسجيل الدخول إلى الأجهزة البعيدة. كان هنالك برنامجا `rlogin` و `telnet`. لكن كان يعاني هذان البرنامجان من خطأ فادح كان قد وقع فيه FTP؛ كانا ينقلان جميع الاتصالات (بما فيها اسم المستخدم وكلمة المرور) على شكل نص دون أي تشفير! وهذا ما جعلهما غير مناسبَين لعصر الإنترنت.

ssh

لحل تلك المشكلة، طُوِّرَ بروتوكول جديد يسمى SSH (Secure Shell). يَحُلّ SSH مشكلتَين أساسيتَين في الاتصال الآمن بحاسوب بعيد. أولًا، هو يتحقق من أن الخادم البعيد هو فعلًا الخادم الحقيقي الذي نريد الاتصال به (ونتجنب بذلك الهجوم المسمى "man in the middle")، وثانيًا، هو يشفر جميع الاتصالات ما بين الخادم المحلي والخادم البعيد.

يتألف SSH من قسمين، خادم SSH يعمل على الحاسوب البعيد "يستمع" إلى الاتصالات في المنفذ ذي الرقم 22، وعميل SSH في الجهاز المحلي للتواصل مع الخادم البعيد.

تستخدم أغلب توزيعات لينُكس SSH عن طريق برمجية OpenSSH من مشروع OpenBSD. بعض التوزيعات تحتوي على العميل والخادم مُثبتَين افتراضيًا (على سبيل المثال، ريدهات)، بينما توفر باقي التوزيعات (كأوبنتو) العميل فقط. للسماح لنظامك باستقبال الاتصالات البعيدة، يجب أن تكون الحزمة OpenSSH-server مثبتة ومُعَدّة وتعمل حاليًا. ويجب أيضًا السماح للاتصالات باستخدام منفذ TCP 22.

تلميحة: إذا لم يكن لديك نظام بعيد للاتصال به وأردت تجربة الأمثلة الآتية، فتأكد من تثبيت الحزمة `OpenSSH-server` على نظامك واستخدم `localhost` بدلًا من اسم المضيف البعيد. في هذه الحالة،

سيُنشئ جهازك اتصالات شبكية مع نفسه.

عميل SSH المُستخدم للاتصال بخوادم SSH البعيد يسمى ssh. للاتصال بخادم بعيد وليكن اسمه remote-sys، فإننا نستخدم عميل ssh على النحو الآتي:

```
[me@linuxbox ~]$ ssh remote-sys
The authenticity of host 'remote-sys (192.168.1.4)' can't be
established.
RSA key fingerprint is
41:ed:7a:df:23:19:bf:3c:a5:17:bc:61:b3:7f:d9:bb.
Are you sure you want to continue connecting (yes/no)?
```

ستظهر هذه الرسالة في أول مرّة تتصل فيها مع الخادم البعيد التي تشير إلى أن الاستيثاق بالخادم البعيد لم يتم بعد. وهذا لأن العميل لم يتصل بخادم SSH المحدد من قبل. للموافقة على الاتصال بالخادم، اطبع الكلمة "yes" في المِحث. سيُسأل المستخدم عن كلمة مروره عندما يتم الاتصال بنجاح:

```
Warning: Permanently added 'remote-sys,192.168.1.4' (RSA) to the list
of known hosts.
me@remote-sys's password:
```

بعد أن تُدخَل كلمة المرور بشكل صحيح، فإننا سنتلقى مِحث صدفة من النظام البعيد:

```
Last login: Sat Aug 30 13:00:48 2008
[me@remote-sys ~]$
```

تستمر جلسة الصدفة البعيدة حتى يُدخِل المستخدم الأمر `exit`، حيث سيُغلَق الاتصال، وسيتم الانتقال إلى الصدفة المحلية عوضًا عنها.

من الممكن أيضًا الاتصال بالأنظمة البعيدة باسم مستخدم مختلف. على سبيل المثال، إذا كان للمستخدم "me" حساب آخر باسم "bob" على الخادم البعيد، فيستطيع me الدخول بحساب bob في الخادم البعيد كما يلي:

```
[me@linuxbox ~]$ ssh bob@remote-sys
bob@remote-sys's password:
Last login: Sat Aug 30 13:03:21 2008
[bob@remote-sys ~]$
```

وكما ذُكِر سابقًا، يتحقق ssh من الاستيثاق بالخادم البعيد. إذا لم يتم الاستيثاق من الخادم البعيد بنجاح،

فستظهر الرسالة الآتية:

```
[me@linuxbox ~]$ ssh remote-sys
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@      WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle
attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
41:ed:7a:df:23:19:bf:3c:a5:17:bc:61:b3:7f:d9:bb.
Please contact your system administrator.
Add correct host key in /home/me/.ssh/known_hosts to get rid of this
message.
Offending key in /home/me/.ssh/known_hosts:1
RSA host key for remote-sys has changed and you have requested strict
checking.
Host key verification failed.
```

ظهرت الرسالة السابقة لأحد احتمالَين: الأول، أن مهاجمًا حاول استخدام هجوم "man-in-the-middle".وهذا أمرٌ نادر، لأن الجميع يعرف أن ssh سيحذر المستخدم من ذلك. أو أن الخادم البعيد قد تغير بشكلٍ أو بآخر؛ على سبيل المثال، أُعيد تثبيت نظام التشغيل أو خادم SSH. ولضمان الأمن والحماية، لا تستبعد الاحتمال الأول. راجع مدير النظام البعيد لمعرفة سبب ظهور الرسالة.

إذا تأكدت أن سبب الرسالة هو تغيير في الخادم، أصبح من الآمن أن نحاول إصلاح المشكلة في طرف العميل. وذلك باستخدام محرر نصي (ربما vim) لإزالة المفتاح القديم من ملف `~/.ssh/known_hosts`. تُظهر الرسالة السطر الآتي:

```
Offending key in /home/me/.ssh/known_hosts:1
```

هذا يعني أن ملف known_hosts يحتوي على المفتاح المخالف في السطر الأول. أزل ذاك السطر وسيستطيع ssh قبول طلب الاستيثاق من الخادم البعيد مرّة أُخرى.

يسمح ssh لنا بتنفيذ أمر واحد في النظام البعيد دون فتح جلسة طرفية كاملة. على سبيل المثال، لتنفيذ الأمر `free` على خادم بعيد باسم `remote-sys` وإظهار النتائج على جهازنا المحلي:

```
[me@linuxbox ~]$ ssh remote-sys free
me@twin4's password:
             total       used     free     shared    buffers     cached
Mem:        775536     507184   268352          0     110068     154596
-/+ buffers/cache:     242520   533016
Swap:      1572856          0  1572856
[me@linuxbox ~]$
```

من الممكن استخدام هذه التقنية بطرق متعددة مفيدة، على سبيل المثال، نريد تنفيذ الأمر `ls` على الخادم البعيد وإعادة توجيه المخرجات إلى ملف في نظامنا المحلي:

```
[me@linuxbox ~]$ ssh remote-sys 'ls *' > dirlist.txt
me@twin4's password:
[me@linuxbox ~]$
```

لاحظ استخدام علامات الاقتباس المفردة في الأمر السابق. قمنا بذلك لأننا لا نريد حدوث توسعة أسماء الملفات في الجهاز المحلي؛ بل نريدها أن تتم على الخادم البعيد. وبشكل مشابه؛ إذا أردنا تحويل المخرجات إلى ملف في الخادم البعيد، فإننا نضع معامل إعادة التوجيه واسم الملف ضمن علامتَيّ الاقتباس المفردتَين:

```
[me@linuxbox ~]$ ssh remote-sys 'ls * > dirlist.txt'
```

إنشاء الأنفاق مع SSH

أحد الأشياء التي تحدث أثناء إنشائك اتصالًا عبر SSH هو إنشاء"نفق مشفر" بين جهازك المحلي والخادم البعيد. يُستخدم هذا النفق لكي تنتقل الأوامر التي نكتبها نقلًا آمنًا إلى الخادم البعيد، ولكي تمر عبره النتائج أيضًا بأمان. بالإضافة إلى ذلك، يدعم بروتوكول SSH مرور أغلب أنواع البيانات عبر النفق، منُشئًا ما يُشبه VPN (اختصار للعبارة "Virtual Private Network") بين النظام المحلي والبعيد.

ربما أشهر استخدام لهذه الميزة هو السماح لبيانات نظام النوافذ X بالمرور عبر النفق. من الممكن تشغيل برنامج رسومي على الخادم وإظهاره على الجهاز المحلي في نظام يُشغّل خادم X (أي أنه يعرض واجهة رسومية). من السهل القيام بذلك، لنفترض أننا على نظام لينُكس يدعى `linuxbox` الذي يُشغّل خادم X، ونريد تشغيل البرنامج `xload` على خادم بعيد اسمه `remote-sys` ومشاهدة محتوى النافذة على جهازنا المحلي، فإننا ننفذ هذا الأمر:

```
[me@linuxbox ~]$ ssh -X remote-sys
```

```
me@remote-sys's password:
Last login: Mon Sep 08 13:23:11 2008
[me@remote-sys ~]$ xload
```

بعد أن يُنفَّذ الأمر xload على الخادم البعيد، فسوف تظهر النافذة على النظام المحلي. في بعض الأنظمة، تحتاج إلى استخدام الخيار "Y-" بدلًا من الخيار "X-".

scp و sftp

تحتوي حزمة OpenSSH أيضًا على برنامجَين يَستخدمان خاصية الأنفاق المشفرة التي يوفرها SSH لنقل الملفات عبر الشبكة. الأول، scp (اختصار للعبارة "secure copy") يُستخدم بشكل مشابه للأمر cp لنسخ الملفات. أكبر الفروق هو أن المسار المنسوخ منه أو إليه يمكن أن يُسبَق باسم الخادم البعيد يتبعه رمز النقطتَين الرأسيتَين. على سبيل المثال، إذا أردنا نسخ ملف يسمى document.txt من مجلد المنزل الخاص بنا في النظام البعيد remote-sys إلى مجلد العمل الحالي في النظام المحلي، فإننا ننفذ:

```
[me@linuxbox ~]$ scp remote-sys:document.txt .
me@remote-sys's password:
document.txt            100% 5581         5.5KB/s      00:00
[me@linuxbox ~]$
```

وكما في ssh، يمكن أن نحدد اسم المستخدم في بداية اسم المضيف البعيد في حال كان اسم المستخدم في ذاك النظام ليس نفسه في النظام المحلي:

```
[me@linuxbox ~]$ scp bob@remote-sys:document.txt .
```

برنامج نسخ الملفات الثاني هو sftp الذي -كما يُبين اسمه- هو بديل آمن لبرنامج ftp. برنامج sftp يعمل بشكل مشابه لبرنامج ftp الذي استخدمناه سابقًا؛ إلا أنه يستخدم نفق آمن بدلًا من إرسال كل شيء بدون تشفير. sftp يحتوي على ميزة تجعله متفوقًا على ftp حيث أنه لا يتطلب وجود خادم FTP في المضيف البعيد. هو يتطلب وجود خادم SSH فقط. هذا يعني أن أي جهاز بعيد يمكن الاتصال به بواسطة عميل SSH يمكن أيضًا استخدامه كخادم شبيه بخادم FTP. هذه جلسة توضيحية:

```
[me@linuxbox ~]$ sftp remote-sys
Connecting to remote-sys...
me@remote-sys's password:
sftp> ls
```

```
ubuntu-8.04-desktop-i386.iso
sftp> lcd Desktop
sftp> get ubuntu-8.04-desktop-i386.iso
Fetching /home/me/ubuntu-8.04-desktop-i386.iso to ubuntu-8.04-
desktop-i386.iso
/home/me/ubuntu-8.04-desktop-i386.iso 100% 699MB
7.4MB/s  01:35
sftp> bye
```

تلميحة: بروتوكول SFTP مدعوم من قِبل العديد من مدراء الملفات الرسوميين الموجودين في توزيعات لينُكس. نستطيع عند استخدام نوتاليز (غنوم) أو كُنكرر (كدي) إدخال مسار يبدأ باللاحقة `sftp://` في شريط المسار وإدارة الملفات الموجودة على مضيف بعيد يُشغل خادم SSH.

عميل SSH لويندوز؟

لنفترض أنك تجلس على جهاز يعمل بنظام تشغيل ويندوز ولكنك تريد أن تسجل الدخول إلى خادم لينُكس الخاص بك وتقوم ببعض الأعمال هناك، ماذا عليك أن تفعل؟ بالطبع أن تحصل على عميل SSH لويندوز! يوجد هنالك عدد منهم. أشهر برنامج هو PuTTY الذي يُظهِر نافذة الطرفية ويسمح لمستخدمي ويندوز بإنشاء جلسات SSH (أو telnet) على المضيف البعيد، يوفر هذا البرنامج أيضًا برنامجَيّ `scp` و `sftp`.

برنامج PuTTY متوفر في:

http://www.chiark.greenend.org.uk/~sgtatham/putty/

الخلاصة

لقد استعرضنا في هذا الفصل مختلف أدوات الشبكة المتوفرة في أغلب أنظمة لينُكس. ولما كان نظام لينُكس منتشرًا انتشارًا كبيرًا في الخوادم والأجهزة الشبكية؛ فيوجد الكثير من الأدوات الإضافية التي نستطيع الحصول عليها بتثبيت بعض البرمجيات. مع ذلك، نستطيع القيام بالعديد من المهام التي تتعلق بالشبكة باستخدام هذه الأدوات الأساسية.

الفصل السابع عشر:

البحث عن الملفات

من المؤكد أنّه اتضحت لنا الحقيقة الآتية أثناء تجولنا في نظام لينُكس: يحتوي نظام لينُكس ملفات كثيرة. وهذا ما يثير التساؤل الآتي: "كيف نستطيع إيجاد الملفات؟". نحن نعلم أن نظام لينُكس منظم تنظيمًا جيدًا وذلك بالاعتماد على معايير ورث بعضها من الجيل الذي يسبقه من الأنظمة الشبيهة بِيونكس. لكن مع هذا، يبقى العدد الضخم من الملفات مربكًا.

سنناقش في هذا الفصل أداتَين تُستخدمان للعثور على الملفات في النظام. هاتان الأداتان هما:

- `locate` – العثور على الملفات حسب الاسم.
- `find` – البحث عن الملفات في شجرة نظام الملفات.

وسنلقي نظرة على أمرٍ يُستخدم بكثرة مع عمليات البحث لمعالجة قائمة الملفات الناتجة:

- `xargs` – بناء وتنفيذ أوامر من مجرى الدخل القياسي.

بالإضافة إلى ذلك، سنستخدم أمرَين لمساعدتنا في الشرح:

- `touch` – تغيير وقت تعديل الملفات.
- `stat` – عرض حالة ملفٍ ما.

العثور على الملفات بالطريقة السهلة باستخدام locate

يجري البرنامج `locate` بحثًا سريعًا جدًا في قاعدة بيانات تحتوي على مسارات جميع الملفات الموجودة في النظام، ويُظهِر المسار الذي تطابق كلمة البحث جزءًا منه. لنفترض، على سبيل المثال، أننا نريد العثور على كل البرامج التي تبدأ بالكلمة "`zip`". ولأننا نبحث عن برامج، فإننا سنعتبر أن المجلد الذي يحتوي على البرنامج ينتهي بالعبارة "`bin/`". لنستخدم الأمر `locate` بهذه الطريقة للعثور على الملفات:

```
[me@linuxbox ~]$ locate bin/zip
```

سيبحث البرنامج `locate` في قاعدة بيانات تحتوي على مسارات الملفات عن أي مسار يحتوي السلسلة النصية "`bin/zip`":

```
/usr/bin/zip
```

```
/usr/bin/zipcloak
/usr/bin/zipgrep
/usr/bin/zipinfo
/usr/bin/zipnote
/usr/bin/zipsplit
```

إذا كانت متطلبات البحث ليست بسيطة جدًا، فيمكن استخدام locate مع أداة أخرى كأداة grep لإنشاء أوامر بحث أكثر تعقيدًا:

```
[me@linuxbox ~]$ locate zip | grep bin
/bin/bunzip2
/bin/bzip2
/bin/bzip2recover
/bin/gunzip
/bin/gzip
/usr/bin/funzip
/usr/bin/gpg-zip
/usr/bin/preunzip
/usr/bin/prezip
/usr/bin/prezip-bin
/usr/bin/unzip
/usr/bin/unzipsfx
/usr/bin/zip
/usr/bin/zipcloak
/usr/bin/zipgrep
/usr/bin/zipinfo
/usr/bin/zipnote
/usr/bin/zipsplit
```

البرنامج locate موجود منذ العديد من السنوات، ويوجد هناك نسخ مختلفة منه. أكثر نسختين استخدامًا في توزيعات لينُكس الحديثة هما slocate و mlocate، اللتان يمكن الوصول إليهما عبر الوصلة الرمزية المسماة locate. تحتوي النسخ المختلفة من locate على مجموعة مختلفة من الخيارات القابلة للاستخدام. تتضمن بعض النسخ إمكانية المطابقة باستخدام التعابير النظامية (التي سنشرحها في فصلٍ لاحق) ودعم المحارف البديلة. راجع صفحة الدليل للأمر locate كي تُحدِّد أيّة نسخة من locate ثُبِّتت على نظامك.

من أين تأتي قاعدة بيانات locate؟

قد تلاحظ أن locate يفشل بالعمل مباشرةً بعد تثبيت النظام في بعض التوزيعات. لكن إذا جرّبت بعد يومٍ واحد، فإنه سيعمل جيدًا. لماذا؟ تُنشأ قاعدة بيانات locate من قِبل برنامجٍ آخر يسمى updatedb. يُشغَّل ذاك البرنامج عادةً كمهمة دورية (cron jobs)؛ أي المهام تُنفَّذ بعد مرور فاصل زمني معين. معظم الأنظمة التي تحتوي على locate تُشغّل updatedb مرّة كل يوم. ولأن قاعدة البيانات لا تُحدَّث تحديثًا مستمرًا، لذا، قد تجد أن بعض الملفات المنشأة حديثًا لا تظهر باستخدام الأمر locate، لتجاوز هذه الإشكالية، من الممكن تشغيل برنامج updatedb يدويًا بتنفيذ الأمر updatedb بامتيازات الجذر.

العثور على الملفات بالطريقة الصعبة باستخدام find

بينما يعثر برنامج locate على الملفات بالاعتماد فقط على اسمها، يبحث برنامج find في مجلد محدد (وجميع المجلدات الفرعية المحتواة فيه) عن الملفات بالاعتماد على قائمة بالخصائص. سنمضي وقتًا طويلًا مع find لأنه يحتوي على الكثير من الميزات المثيرة للاهتمام التي سنشاهدها مرّة ثانية عندما نبدأ بشرح مواضيع البرمجة في الفصول الأخيرة.

بأبسط استخداماته، يُمرَر إلى find اسم مجلد أو أكثر للبحث فيها. مثلًا، لإنشاء قائمة تحتوي على جميع محتويات مجلد المنزل الخاص بنا:

```
[me@linuxbox ~]$ find ~
```

سيولد الأمر السابق، في أغلب حسابات المستخدمين، قائمةً كبيرةً جدًا. ولأن القائمة ستُرسَل إلى مجرى الخرج القياسي، فنستطيع تحويلها عبر الأنبوب إلى برامج أُخرى. لنستخدم الأمر wc لإحصاء عدد الملفات:

```
[me@linuxbox ~]$ find ~ | wc -l
47068
```

عدد ضخم من الملفات! إحدى ميزات find الرائعة هي قابلية استخدامه لتحديد الملفات التي تطابق معيارًا أو مقياسًا محددًا؛ وذلك باستخدام "الخيارات" (options) و"الاختبارات" (tests) و"الأفعال" (actions). سنلقي أولًا نظرة على الاختبارات.

الاختبارات

لنفترض أننا نريد قائمةً بالمجلدات الموجودة في مكان البحث. للقيام بذلك، نستخدم الاختبار الآتي:

```
[me@linuxbox ~]$ find ~ -type d | wc -l
1695
```

إضافة الاختبار `type d-` يجعل البحث مقتصرًا على المجلدات فقط. بشكل مشابه، نستطيع أن نجعل البحث مقتصرًا على الملفات العادية:

```
[me@linuxbox ~]$ find ~ -type f | wc -l
38737
```

يحتوي الجدول الآتي على قائمة بأبرز اختبارات الملفات المدعومة من قِبل الأمر `find`:

الجدول 17-1: أنواع الملفات المُعتمَدة من قِبل find

رمز الملف	نوع الملف
b	ملف جهاز كتلي (`block`) خاص.
c	ملف جهاز حرفي (`Character`) خاص.
d	مجلد.
f	ملف عادي.
l	وصلة رمزية.

يمكننا أيضًا البحث بتحديد الحجم التخزيني للملف واسم الملف؛ وذلك باستخدام المزيد من الاختبارات. لنبحث عن جميع الملفات العادية التي تحتوي على النمط "`*.JPG`" وحجمها التخزيني أكبر من واحد ميغابايت:

```
[me@linuxbox ~]$ find ~ -type f -name "*.JPG" -size +1M | wc -l
840
```

أضفنا، في المثال السابق، الاختبار name- متبوعًا بنمط يحتوي على محارف بديلة. لاحظ كيف استخدمنا علامتَيّ الاقتباس المزدوجتَين لكي نمنع الصدفة من توسيع أسماء الملفات. ومن ثم استخدمنا الاختبار `size-` متبوعًا بالسلسلة النصية "+1M". إشارة الموجب تعني أننا نبحث عن ملفات أكبر من الرقم المحدد. إذا

استخدمنا إشارة السالب، فهذا يعني أننا نريد البحث عن ملفات أصغر من الرقم المحدد. وعند عدم وضع إشارة فهذا يعني أننا نريد أن يكون حجم الملف مساويًا للرقم المحدد. الحرف "M" يبين أن واحدة القياس المُستخدمة هي الميغابايت. يمكن استخدام الأحرف الآتية لتحديد الواحدات:

الجدول 17-2: واحدات الحجم التخزيني المُعتمَدة من قِبل find

الحرف	الشرح
b	كتل 512 بايت. الخيار الافتراضي إن لم تُحدَّد الواحدة.
c	بايت.
w	كلمات 2-بايت.
k	كيلوبايت (1024 بايت).
M	ميغابايت (1048576 بايت).
G	غيغابايت (1073741824 بايت).

يدعم الأمر find عددًا كبيرًا من الاختبارات المختلفة. يسرد الجدول الآتي أبرزها. لاحظ أنه سيكون لإشارتَيّ "+" و "-" نفس التأثير الذي شرحناه سابقًا في الحالات التي تُستخدم فيها الأرقام كوسيط.

الجدول 17-3: اختبارات find

الاختبار	الشرح
-cmin n	مطابقة الملف أو المجلد الذي عُدِّلَ محتواه أو خصائصه (attributes أي الأذونات أو وقت التعديل ...إلخ.) منذ n دقيقة. لتحديد المدة بأقل من n دقيقة، استخدم n-. ولتحديد المدة بأكثر من n دقيقة استخدم n+.
-cnewer file	مطابقة الملفات أو المجلدات التي عُدِّلَ محتواها أو خصائصها بعد الملف file.
-ctime n	مطابقة الملفات أو المجلدات التي عُدِّلَ محتواها أو خصائصها منذ 24 * n ساعة.
-empty	مطابقة الملفات والمجلدات الفارغة.
-group name	مطابقة الملفات أو المجلدات التي تتعلق بالمجموعة group. يمكن أن يُعبَّر عن المجموعة إما بكتابة اسمها أو بمُعرِّف المجموعة.

`-iname pattern`	كالاختبار name- لكنه غير حساس لحالة الأحرف.
`-inum n`	مطابقة الملفات ذات رقم العقدة n. هذا الاختبار مفيد للعثور على جميع الوصلات الصلبة لعقدة معينة.
`-mmin n`	مطابقة الملفات أو المجلدات التي غُيّرَ محتواها منذ n دقيقة.
`-mtime n`	مطابقة الملفات أو المجلدات التي غُيّرَ محتواها منذ n*24 ساعة.
`-name pattern`	مطابقة الملفات أو المجلدات التي يطابق اسمها النمط `pattern`.
`-newer file`	مطابقة الملفات والمجلدات التي عُدِّلت محتوياتها بعد الملف `file`. هذا الاختبار مفيد جدًا عند كتابة سكربتات شِل التي تُنشئ نسخًا احتياطيةً من الملفات. كل مرّة تأخذ فيها نسخة احتياطية أو يتم تحديث ملف (كملفات السجلات `logs`)، فاستخدم الأمر `find` لكي تحدد أيّة ملفات قد عُدِّلت منذ التحديث الأخير.
`-nouser`	مطابقة الملفات والمجلدات التي لا تنتمي لحساب مستخدم ساري المفعول. يمكن استخدام هذا الاختبار لمطابقة الملفات التي تنتمي لمُستخدمين قد حُذفت حساباتهم أو لاكتشاف نشاط المهاجمين.
`-nogroup`	مطابقة الملفات والمجلدات التي لا تنتمي لمجموعة سارية المفعول.
`-perm mode`	مطابقة الملفات أو المجلدات التي ضُبِطَت الأذونات الخاصة بها إلى mode. يمكن التعبير عن mode كقيم رمزية أو أرقام بالنظام الثماني.
`-samefile name`	مشابه للاختبار `inum-`. يطابق الملفات التي تتشارك بنفس رقم العقدة.
`-size n`	مطابقة الملفات ذات الحجم التخزيني n.
`-type c`	مطابقة الملفات ذات النوع c.
`-user name`	مطابقة الملفات أو المجلدات التي يملكها المستخدم name. يمكن التعبير عن المستخدم بتحديد اسمه أو رقم ID الخاص به.

هذه ليست قائمةً كاملةً بجميع الاختبارات. راجع صفحة الدليل للأمر `find` لمزيدٍ من التفاصيل.

المعاملات

على الرغم من وجود عددٍ كبير من الخيارات للأمر find، لكننا نحتاج إلى طريقة لتحديد العلاقات المنطقية بين الاختبارات. على سبيل المثال، ماذا لو أردنا العثور على جميع الملفات والمجلدات الفرعية التي تكون أذوناتها "آمنة"؟ سنبحث عن جميع الملفات ذات الأذونات التي لا تساوي 0600 وجميع المجلدات ذات الأذونات التي لا تساوي 0700. لحسن الحظ، يوفر find طريقةً لدمج الاختبارات عن طريق المعاملات المنطقية (logical operators) لإنشاء علاقات منطقية أكثر تعقيدًا. لإنشاء الاختبار السابق، نستخدم الأمر:

```
[me@linuxbox ~]$ find ~ \( -type f -not -perm 0600 \) -or \( -type d
-not -perm 0700 \)
```

ما كل هذه الرموز!؟ ليست المعاملات معقدةً إذا تعلمتَ طريقة عملهم:

الجدول 4-17: معاملات find المنطقية

المعامل	الشرح
-and	يطابق الملف في حال كان الاختبار على طرفي المعامل محققًا. يمكن اختصاره إلى a-. لاحظ أنه في حال عدم تحديد معامل ما، فسيستخدم and- افتراضيًا.
-or	يطابق الملف في حال كان أحد الاختبارَين على جانبي المعامل محققًا. يمكن اختصاره إلى o-.
-not	يطابق الملف في حال كان الاختبار الذي يلي المعامل غير محقق. يمكن اختصار هذا المعامل إلى إشارة التعجب "!".
( )	إنشاء مجموعة من الاختبارات أو المعاملات لتكوين تعابير أكبر. يستخدم للتحكم في الترتيب المنطقي للتعابير. افتراضيًا، يأخذ find الاختبارات بعين الاعتبار من اليسار إلى اليمين. يكون عادةً من الضروري أن يُستبدل السلوك الافتراضي للحصول على النتائج المطلوبة. حتى وإن لم يكن استخدام الأقواس ضروريًا، فقد يزيد من سهولة قراءة الأمر. ولأن للأقواس معنى خاص في الصدفة، فلاحظ أننا قمنا "باقتباسهم" للسماح بتمريرهم كوسائط إلى find؛ وذلك باستخدام الشرطة المائلة الخلفية.

بعد التعرف على قائمة المعاملات، لنتفحص أمر find السابق بدقة. يمكن أن يُنظر إليه على أنه مجموعتان من الاختبارات يفصل بينهما المعامل "or-":

```
( expression 1 ) -or ( expression 2 )
```

إذا كنا نبحث عن الملفات والمجلدات، فلماذا أُستخدم or- بدلًا من and-؟ لأنه عندما يبحث find عن الملفات والمجلدات، فإنه يقارنها بأحد التعبيرين وليس كليهما. نحن نريد أن نعرف فيما إن كان ملفًا ذا أذونات "سيئة" أو مجلدًا ذا أذونات سيئة. لا يمكن أن يكون الاثنان سويةً:

```
( file with bad perms ) -or ( directory with bad perms )
```

الآن يجب علينا معرفة كيف نحدد "الأذونات السيئة"؟ في الواقع، نحن لا نبحث عن الأذونات السيئة بل عن "أذونات ليست جيدة". في حالتنا هذا، نُعرِّف الأذونات الجيدة للملفات على أنها 0600 و 0700 للمجلدات. التعبير الذي يختبر أذونات الملفات هو:

```
-type f -and -not -perms 0600
```

وتعبير المجلدات:

```
-type d -and -not -perms 0700
```

وكما ذكرنا في جدول المعاملات، يمكن حذف المعامل and- لأنه يُستخدم افتراضيًّا. لذا، إذا جمعنا جميع المعلومات السابقة، فإننا نحصل على الأمر الآتي:

```
find ~ ( -type f -not -perms 0600 ) -or ( -type d -not -perms 0700 )
```

لكن ولما كان للأقواس معنى خاص بالنسبة إلى الصدفة، فيجب علينا تهريبهم لمنع الصدفة من محاولة تفسيرهم. يتم ذلك بإضافة شرطة مائلة خلفية قبل كل قوس.

يوجد هنالك ميزة أخرى مهمة من مزايا المعاملات المنطقية من المهم معرفتها. لنفترض أنه لدينا تعبيران يفصل بينهما معامل منطقي:

```
expr1 -operator expr2
```

في جميع الحالات يُنفّذ التعبير الأول expr1؛ لكن المعامل هو الذي يُحدِّد فيما إن كان سيُنفَّذ التعبير الثاني expr2:

الجدول 17-5: توضيح استخدام AND/OR في find

نتيجة التعبير expr1	المعامل	هل يتم تنفيذ التعبير expr2؟
true	-and	يُنفذ دائمًا
false	-and	لا يُنفذ أبدًا
true	-or	لا يُنفذ أبدًا
false	-or	يُنفذ دائمًا

يتـم ذلـك لتحسـين الأداء. لنفـرض أن المعامـل هـو and- علـى سـبيل المثـال. نحـن نعلـم أن expr1 -and expr2 لا يمكن أن يكون محققًا إذا كان التعبير expr1 غير محقق، لذا لن يكون هنالك طائل من تنفيذ التعبير expr2. بشكل مشابه، إذا كان expr1 -or expr2 وكان التعبير expr1 محققًا، فلا فائدة من تنفيذ expr2. لأننا نعلم مسبقًا أن التعبير expr1 -or expr2 محقق.

حسنًا، هذه الميزة تساعد في تحسين الأداء، لكن لماذا هي مهمة للغاية!؟ الأهمية تكمن في أننا نستطيع الاعتماد على هذا السلوك للتحكم في كيفية تنفيذ الأفعال (actions).

تنفيذ الأفعال

لننفذ الآن بعض الأعمال! الحصول على قائمة بالملفات التي عثر عليه الأمر find هو شيء مفيد، لكن ما نريده فعلًا هو تنفيذ بعض الأفعال على عناصر تلك القائمة. لحسن الحظ، يسمح find بأن تُنفَّذ الأوامر بناءً على نتائج البحث. يوجد هنالك أفعال معرّفة مسبقًا وعدّة طرق لتنفيذ أفعال تُعرَّف من قِبل المستخدم. لنلقِ أولًا نظرة على الأفعال المُعرَّفة مُسبقًا:

الجدول 17-6: أفعال find المُعرفة مُسبقًا

الفعل	الشرح
-delete	حذف الملفات المطابقة.
-ls	تنفيذ الأمر ls -dils على الملفات التي تمت مطابقتها وإرسال المخرجات إلى مجرى الخرج القياسي.
-print	طباعة المسار الكامل للملفات التي تمت مطابقتها إلى مجرى الخرج القياسي. هذا هو الفعل الافتراضي في حال عدم تحديد أي فعل آخر.
-quit	الخروج عند مطابقة أحد النتائج.

وكما الاختبارات، يوجد هنالك العديد من الأفعال. راجع صفحة الدليل للأمر find لمزيد من التفاصيل.

في أول مثال لنا، نفذنا الأمر:

```
find ~
```

الذي أظهر قائمةً بجميع الملفات والمجلدات الفرعية الموجودة داخل مجلد المنزل. أنشأ الأمر السابق قائمةً لأن الفعل print- أُستخدم نتيجة عدم تحديد أي فعل آخر. لذا، يمكن التعبير عن الأمر السابق كالآتي:

```
find ~ -print
```

يمكننا استخدام الأمر find لحذف الملفات التي تُطابق شروطًا معينةً. على سبيل المثال، لحذف جميع الملفات التي تنتهي بالامتداد ".BAK" (عادةً ما يُستخدم هذا الامتداد للإشارة إلى ملفات النسخ الاحتياطي)، يمكننا استخدام الأمر الآتي:

```
find ~ -type f -name '*.BAK' -delete
```

يبحث الأمر السابق بين جميع الملفات الموجودة في مجلد المنزل للمستخدم (وجميع المجلدات الفرعية) عن اسم ملف ينتهي بالامتداد BAK. ومن ثم يحذف تلك الملفات.

تحذير: لا يمكن أن يمرّ الموضوع دون تنبيهك أن تأخذ أقصى درجات الحذر عند استخدام الفعل ‎-delete. جرّب دائمًا الأمر باستخدام الفعل ‎-print للتأكد من صحة النتائج.

قبل أن نُكمل، علينا أن نلقي نظرة حول طريقة تأثير المعاملات المنطقية على الأفعال. خذ على سبيل المثال الأمر الآتي:

```
find ~ -type f -name '*.BAK' -print
```

كما لاحظنا، يبحث الأمر السابق عن الملفات العادية (‎-type f) التي تنتهي أسماؤها بالامتداد BAK. (‎-name '*.BAK') وسنطبع المسارات النسبية لكل ملف تتم مطابقته إلى مجرى الخرج القياسي (‎-print). لكن تنفيذ الأمر تنفيذًا صحيحًا يكون محددًا وفق العلاقات المنطقية بين كلٍ من الاختبارات والأفعال. يمكننا التعبير عن الأمر السابق بالطريقة الآتية لجعل قراءته أسهل:

```
find ~ -type f -and -name '*.BAK' -and -print
```

لنلقِ الآن نظرة حول طريقة تأثير المعاملات المنطقية على التنفيذ:

الفعل	سيُنفّذ الاختبار إذا كان
‎-print	‎-type f و‎-name '*.BAK' محققيَن.
‎-name‘*.BAK’	‎-type f محققًا.
‎-type f	يُنفَّذ دائمًا، لأنه التعبير الأول في عبارة ‎-and.

ولما كانت العلاقة المنطقية بين الاختبارات والأفعال تُحدد أيًّا منهم سينفذ، فيمكننا ملاحظة أن ترتيب

الاختبارات والأفعال مهم جدًا. على سبيل المثال، إذا غيرّنا ترتيب الأمر السابق ووضعنا print- في أول الأمر، فسيتغير سلوك find تمامًا:

```
find ~ -print -and -type f -and -name '*.BAK'
```

سيطبع الأمر السابق جميع الملفات والمجلدات ومن ثم سيختبر نوع الملف وامتداده.

الأفعال التي تُعرَّف من قِبل المستخدم

بالإضافة إلى الأفعال المعرّفة مسبقًا، يُمكننا صنع أوامر خاصة بنا. الطريقة التقليدية لفعل ذلك هي الفعل exec-. يعمل ذاك الفعل بالطريقة الآتية:

```
-exec command {} ;
```

حيث command هو اسم الأمر، و {} هو رمز يشير إلى مسار الملف، وتوضع الفاصلة المنقوطة ";" للإشارة إلى نهاية الأمر. هذا مثال عن استخدام exec- للقيام بنفس العمل الذي يقوم به الفعل "delete-":

```
-exec rm '{}' ';'
```

يجب علينا تهريب القوسَين والفاصلة المنقوطة لأنهما محرفان ذوا معنى خاص.

من الممكن أيضًا أن تُنفَّذ الأفعال التي عرّفها المُستخدم تفاعليًّا؛ وذلك باستخدام الفعل ok- بدلًا من exec-. سيُسأل المستخدم قبل تنفيذ كل أمر:

```
find ~ -type f -name 'foo*' -ok ls -l '{}' ';'
< ls ... /home/me/bin/foo > ? y
-rwxr-xr-x 1 me me 224 2007-10-29 18:44 /home/me/bin/foo
< ls ... /home/me/foo.txt > ? y
-rw-r--r-- 1 me me   0 2008-09-19 12:53 /home/me/foo.txt
```

بحثنا في المثال السابق عن الملفات التي يبدأ اسمها بالعبارة "foo" ومن ثم نفذنا الأمر ls -l في كل مرّة نجد فيها أحد تلك الملفات.

زيادة السرعة والفعّالية

عند تنفيذ الفعل exec-، فسننفذ الأمر المحدد في كل مرّة نعثر فيها على ملف. في بعض الحالات يكون من الأفضل جمع جميع نتائج البحث وتنفيذ أمر واحد فقط. على سبيل المثال، عوضًا عن تنفيذ الأوامر بالشكل:

```
ls -l file1
```

```
ls -l file2
```

يُفضَّل تنفيذها بالشكل:

```
ls -l file1 file2
```

وهذا ما يؤدي إلى تنفيذ الأمر مرة واحدة عوضًا عن تنفيذه لمراتٍ متعددة. توجد طريقتان للقيام بذلك. الطريقة التقليدية: استخدام الأمر xargs، والطريقة البديلة: استخدام ميزة جديدة في الأمر find نفسه. سنتحدث عن الطريقة البديلة أولًا.

بتغيير الفاصلة المنقوطة إلى إشارة الزائد، فإننا نُفعِّل إمكانية تمرير جميع نتائج البحث كقائمة وسائط للأمر المحدد الذي سيُنفَّذ لمرة واحدة. بالعودة إلى مثالنا السابق، فإن الأمر:

```
find ~ -type f -name 'foo*' -exec ls -l '{}' ';'
-rw-r-xr-x 1 me me 244  2007-10-29  18:44 /home/me/bin/foo
-rw-r--r-- 1 me me   0  2008-09-19  18:44 /home/me/foo.txt
```

سينفذ الأمر ls كل مرّة يُطابَق أحد الملفات فيها. لكن عند تغيير الأمر إلى:

```
find ~ -type f -name 'foo*' -exec ls -l '{}' +
-rw-r-xr-x 1 me me 244  2007-10-29  18:44 /home/me/bin/foo
-rw-r--r-- 1 me me   0  2008-09-19  18:44 /home/me/foo.txt
```

سنحصل على نفس النتائج، لكن سينفذ النظام الأمر ls مرّة واحدة فقط.

xargs

يقبل الأمر xargs المدخلات من مجرى الدخل القياسي ويحولها إلى قائمة وسائط للأمر المحدد. سيكون الأمر السابق بهذا الشكل عند استخدام xargs:

```
find ~ -type f -name 'foo*' -print | xargs ls -l
-rw-r-xr-x 1 me me 244  2007-10-29  18:44 /home/me/bin/foo
-rw-r--r-- 1 me me   0  2008-09-19  18:44 /home/me/foo.txt
```

تُمرَّر مخرجات الأمر find إلى xargs الذي بدوره يُنشئ قائمة وسائط للأمر ls ومن ثم ينفذها.

ملاحظة: على الرغم من أن عدد الوسائط التي يمكن أن يقبلها أحد الأوامر كبير جدًا، لكنه محدود. من الممكن إنشاء أوامر طويلة جدًا بحيث لا يمكن للصدفة قبولها. عندما يتم تجاوز الحد الأقصى المحدد من

قِبل النظام، فينفذ xargs الأمر المحدد بأقصى حد من الوسائط ومن ثم يكرر العملية إلى أن ينتهي من جميع الوسائط. لمعرفة الحد الأعلى لطول الأوامر، نفذ xargs مع الخيار show-limits--.

التعامل مع أسماء الملفات "المضحكة"!

تسمح الأنظمة الشبيهة بِيونكس بتضمين فراغات (أو حتى أسطر جديدة) في أسماء الملفات. وهذا ما يسبب مشاكل لبعض البرامج كبرنامج xargs الذي يُنشئ قائمة وسائط للبرامج الأُخرى؛ حيث سيعامل الفراغ كفاصل، وهذا ما يؤدي إلى تفسير الكلمات التي يفصل بينها فراغ على أنها وسائط منفصلة. للتغلب على هذه الإشكالية، يَسمح الأمران find و xargs باستخدام (اختياري) لمحرف "اللاشيء" (null character) كفاصل بين الوسائط. يُعرَّف محرف اللاشيء في ASCII على أنه المحرف المُمثَّل بالرقم صفر (على النقيض من الفراغ، الذي يُمثَّل في ASCII بالرقم 32). يدعم الأمر find الفعل print0- الذي يولّد قائمةً مفصولٌ عناصرها بمحرف اللاشيء. ويوجد أيضًا الخيار null-- لبرنامج xargs الذي يقبل مدخلاتٍ مفصولةً بمحرف اللاشيء. مثال:

```
find ~ -iname '*.jpg' -print0 | xargs --null ls -l
```

لقد تحققنا، عند استخدام هذه الطريقة، من أنّ جميع الملفات ستُعالَج معالجةً صحيحةً بما فيها الملفات التي تحتوي أسماؤها على فراغات.

عودة إلى ساحة التجارب

حان الوقت لاستخدام find استخدامًا عمليًا. لنُنشئ مكانًا للتجارب (شبيهًا بذاك الذي أنشأناه في فصلٍ سابق) ولنجرّب بعض ما تعلمناه.

لنُنشئ أولًا مكانًا للتجارب مع الكثير من المجلدات الفرعية والملفات:

```
[me@linuxbox ~]$ mkdir -p playground/dir-{00{1..9},0{10..99},100}
[me@linuxbox ~]$ touch playground/dir-{00{1..9},0{10..99},100}/file-
{A..Z}
```

عجيبة هي قوة سطر الأوامر! بسطرين فقط، أنشأنا مكانًا للتجارب يحتوي على مائة مجلد فرعي وكل مجلد يحتوي على ستة وعشرين ملفًا فارغًا. جرّب القيام بذلك في الواجهة الرسومية إن استطعت!

الطريقة التي استخدمناها هي الأمر المعروف mkdir وخاصية توسعة الأقواس في الصدفة، بالإضافة إلى أمرٍ جديد هو "touch". باستخدام الأمر mkdir مع الخيار p- (الذي يجعل الأمر mkdir يُنشئ المجلدات

الأب للمسارات المحددة) مع توسعة الأقواس، فإننا تمكّنا من إنشاء مائة مجلد.

يستخدم الأمر touch عادةً لتحديد أو تحديث أوقات الدخول والتغيير والتعديل. لكن إذا لم يكن اسم الملف الممرر موجودًا؛ فسيُنشأ ملف فارغ بنفس الاسم.

أنشأنا مائة ملف باسم "file-A" في ساحة التجارب، لنحاول العثور عليهم:

```
[me@linuxbox ~]$ find playground -type f -name 'file-A'
```

لاحظ، وعلى النقيض من ls، أن find لا يرتّب النتائج. الترتيب يحدد من قِبل تصميم قرص التخزين. سننفذ الأمر الآتي للتأكد من وجود مائة ملف:

```
[me@linuxbox ~]$ find playground -type f -name 'file-A' | wc -l
100
```

لنجرّب الآن البحث عن الملفات حسب وقت التعديل. يفيد ذلك عند القيام بالنسخ الاحتياطي أو لترتيب الملفات بترتيب زمني. سنُنشئ أولًا ملفًا مرجعيًا الذي سنقارن أوقات تعديل الملفات بوقت تعديله:

```
[me@linuxbox ~]$ touch playground/timestamp
```

يُنشئ الأمر السابق ملفًا فارغًا باسم timestamp ويضبط وقت التعديل الخاص به إلى الوقت الحالي. يمكننا التحقق من ذلك بالأمر stat، الذي يمكن اعتباره نسخةً مطورةً من ls. يكشف الأمر stat عن جميع المعلومات التي يعرفها النظام عن الملف وخصائصه:

```
[me@linuxbox ~]$ stat playground/timestamp
     File: `playground/timestamp'
     Size: 0 Blocks: 0 IO Block: 4096 regular empty file
Device: 803h/2051d Inode: 14265061 Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/ me)  Gid: ( 1001/ me)
Access: 2008-10-08 15:15:39.000000000 -0400
Modify: 2008-10-08 15:15:39.000000000 -0400
Change: 2008-10-08 15:15:39.000000000 -0400
```

إذا نفذنا الأمر touch على الملف مرّة أخرى، فسنجد أن الأوقات قد تغيرت:

```
[me@linuxbox ~]$ touch playground/timestamp
[me@linuxbox ~]$ stat playground/timestamp
     File: `playground/timestamp'
```

```
    Size: 0 Blocks: 0 IO Block: 4096 regular empty file
Device: 803h/2051d Inode: 14265061 Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/ me)  Gid: ( 1001/ me)
Access: 2008-10-08 15:23:33.000000000 -0400
Modify: 2008-10-08 15:23:33.000000000 -0400
Change: 2008-10-08 15:23:33.000000000 -0400
```

لنستخدم الأمر find لتحديث وقت تعديل بعض الملفات في ساحة التجارب:

```
[me@linuxbox ~]$ find playground -type f -name 'file-B' -exec touch
'{}' ';'
```

يُحدِّث الأمر السابق الوقت لجميع الملفات الموجودة في ساحة التجارب المسماة file-B. سنستخدم الآن الأمر find للتعرف على الملفات المُحدَّثة بمقارنة جميع الملفات بالملف المرجعي timestamp:

```
[me@linuxbox ~]$ find playground -type f -newer playground/timestamp
```

تحتوي القائمة الناتجة على مائة ملف باسم file-B. لأننا حدثنا الملفات بعد إنشاء الملف timestamp، تلك الملفات هي "أحدث" من ملف timestamp وهذا ما يمكن معرفته من الاختبار newer-.

أخيرًا، لنعود إلى مثال الأذونات السيئة الذي أنشأناه سابقًا ولنجربه على playground:

```
[me@linuxbox ~]$ find playground \( -type f -not -perm 0600 \) -or \(
-type d -not -perm 0700 \)
```

سيعرض الأمر السابق مائة مجلد و 2600 ملف (بما فيها ملف timestamp ومجلد playground نفسه، بحاصل 2702) لأن أيّ ملف أو مجلد منهم لا يطابق تعريفنا "للأذونات الجيدة". وحسب معرفتنا بالمعاملات والأفعال، يمكننا إضافة أفعال إلى الأمر السابق لكي نطبق الأذونات الجديدة على الملفات والمجلدات (كلٌّ على حدة):

```
[me@linuxbox ~]$ find playground \( -type f -not -perm 0600 -exec
chmod 0600 '{}' ';' \) -or \( -type d -not -perm 0711 -exec chmod
0700 '{}' ';' \)
```

ربما نجد مع الأيام أنه من الأسهل تنفيذ أمرَين: واحد للملفات وآخر للمجلدات، بدلًا من أمر واحد طويل، لكن من الجيد معرفة طريقة إنشاء مثل تلك الأوامر. الفكرة الأساسية هي إيجاد طريقة للجمع بين المعاملات والأفعال للقيام بمهمات مفيدة.

الخيارات

تُستخدم الخيارات لتحديد مجال بحث الأمر `find`. يمكن أيضًا تضمينهم مع الاختبارات والأفعال عند إنشاء تعبيرات `find`. يحتوي الجدول الآتي على قائمة بأشهر تلك الخيارات:

الجدول 17-7: خيارات find

الخيار	النتيجة
`-depth`	جعل البرنامج `find` يتعامل مع محتويات المجلد قبل المجلد نفسه. يُطبَّق هذا الخيار تلقائيًا عند استخدام الفعل `-delete`.
`-maxdepth levels`	تحديد الرقم الأعظمي لعدد المراحل التي سيبحث فيها `find` في المجلد (أي عدد المجلدات الفرعية). عندما يُنفِّذ الاختبارات والأفعال.
`-mindepth levels`	تحديد العدد الأدنى من المراحل التي سيبحث فيها `find` قبل تنفيذ الاختبارات والأفعال.
`-mount`	جعل `find` يبحث في المجلدات الموصولة لأنظمة ملفات أُخرى.
`-noleaf`	جعل `find` لا يُحسّن أداء البحث بالاعتماد على أنه يبحث في نظام ملفات لينُكس. يُستخدم هذا الخيار عند البحث في أنظمة ملفات ويندوز وأقراص `CD-ROM`.

الخلاصة

من المؤكد انك لاحظتَ أنّ البرنامج `locate` سهل وبسيط مقارنةً مع `find` المُعقّد. توجد استخدامات لكلا البرنامجَين. خذ وقتك لاستكشاف المزايا الكثيرة التي يتمتع فيها `find`. يمكنه أن يوسّع لك معرفتك بعمليات أنظمة الملفات.

الفصل الثامن عشر:

الأرشفة والنسخ الاحتياطي

إحدى المهمات الأساسية لمدير النظام هي إبقاء البيانات الموجودة في النظام آمنةً. إحدى الطرق لفعل ذلك هي القيام بعمليات نسخ احتياطي وفق جدول زمني محدد. حتى وإن لم تكن مديرًا للنظام، فقد يكون من المفيد عادةً أن تُنشئ نسخًا احتياطيةً من الملفات وتنقل مجموعةً كبيرةً من الملفات من مكانٍ إلى آخر ومن جهازٍ إلى آخر.

سنستعرض في هذا الفصل عددًا من البرامج الشهيرة التي تُستخدم لإدارة مجموعات الملفات. من بينهم برامج ضغط الملفات:

- `gzip` – ضغط أو فك ضغط الملفات.
- `bzip2` – ضاغط ملفات يعتمد على ترتيب الكتل (block sorting).

وبرامج الأرشفة:

- `tar` – إنشاء أرشيفات `tar`.
- `zip` – تحزيم وضغط الملفات.

وبرنامج مزامنة الملفات:

- `rsync` – مزامن للملفات والمجلدات البعيدة.

ضغط الملفات

عبر تاريخ الحوسبة، كان هنالك نضال طويل لتمكين تخزين أكبر كمية من البيانات في أصغر مساحة تخزينية ممكنة، بغض النظر عن كون المساحة التخزينية هي من الذاكرة، أو أجهزة التخزين، أو حتى التراسل الشبكي. العديد من الخدمات الشائعة حاليًا، كالتلفاز العالي الدقة، أو خدمة الإنترنت، أو غيرها الكثير؛ لم يكونوا موجودين لولا تقنيات ضغط البيانات الفعالة.

ضغط البيانات هي عملية إزالة الأجزاء المتكررة من البيانات. لنفرض المثال الآتي: لو كان لدينا صورة بأبعاد مائة بكسل × مائة بكسل. في مصطلحات تخزين البيانات (باعتبار أن كل بكسل يُمثَّل بأربعٍ وعشرين بتًا أو ثلاثة بايتات)، ستحتل الصورة ثلاثين ألفًا من البايتات:

```
100 * 100 * 3 = 30000
```

الصورة التي تحتوي على لون واحد تتكون بشكل شبه كامل من بيانات متكررة. يمكننا ترميز البيانات

بطريقة ما تجعل من السهل وصف أن لدينا صورة تحتوي على 10000 بكسل يحتوي على اللون الأسود. لذا، بدلًا من تخزين البيانات على شكل كتلة تحتوي على ثلاثين ألف صفر (يُمثّل الأسود في الصور بالرقم 0)، يمكننا ضغط البيانات إلى الرقم 10000 متبوعًا بالرقم 0 للتعبير عن البيانات. تسمى هذه التقنية run-length encoding وهي من أشهر تقنيات الضغط عن طريق إزالة التكرارات. التقنيات المستخدمة اليوم هي أكثر تعقيدًا لكنها تسعى إلى الوصول إلى نفس الهدف: إزالة البيانات المتكررة.

تنقسم خوارزميات الضغط (التقنيات الرياضية التي تُستخدم للقيام بعملية الضغط) إلى قسمين عامَين: لا تسبب فقدان البيانات (lossless)، وتسبّب فقدان البيانات (lossy). يحافظ الضغط الذي لا يتسبب بفقدان البيانات على البيانات الموجودة في الملف الأصلي. هذا يعني أنه عندما نستعيد الملف من النسخة المضغوطة؛ فسينتج عندنا نفس النسخة الأصلية دون فقدان أي شيء. أما الضغط الذي يسبب فقدان البيانات فيحذف جزءًا من البيانات عند ضغط الملف للسماح بمزيد من الضغط عليه. عندما يُفَك ضغط الملف، فلن يطابق الملف الأصلي وإنما سيشابهه فقط. الأمثلة هي صيغة JPEG (للصور) و MP3 (للمقاطع الصوتية). سنركز في نقاشنا على الضغط الذي لا يسبب فقدان البيانات، لأن أغلب البيانات على الحاسوب لا يمكن استخدام الضغط الذي يسبب فقدان البيانات معها.

gzip

يُستخدم برنامج `gzip` لضغط ملف واحد أو أكثر. عندما يُنفَّذ، فإنه سيستبدل الملف الأصلي بالنسخة المضغوطة. يُستخدم البرنامج المتمّم `gunzip` لاستعادة الملفات المضغوطة إلى شكلها الأصلي غير المضغوط. هذا مثال عنه:

```
[me@linuxbox ~]$ ls -l /etc > foo.txt
[me@linuxbox ~]$ ls -l foo.*
-rw-r--r-- 1 me me 15738 2008-10-14 07:15 foo.txt
[me@linuxbox ~]$ gzip foo.txt
[me@linuxbox ~]$ ls -l foo.*
-rw-r--r-- 1 me me  3230 2008-10-14 07:15 foo.txt.gz
[me@linuxbox ~]$ gunzip foo.txt
[me@linuxbox ~]$ ls -l foo.*
-rw-r--r-- 1 me me 15738 2008-10-14 07:15 foo.txt
```

أنشأنا، في المثال السابق، ملفًا باسم `foo.txt` ومن ثم نفذنا الأمر `gzip` الذي يستبدل الملف الأصلي بنسخته المضغوطة باسم `foo.txt.gz`. لاحظنا في ناتج أمر "`ls foo*`" أن الحجم التخزيني للملف المضغوط حوالي خُمس حجم الملف الأصلي. لاحظنا أيضًا أن لدى الملف المضغوط نفس الأذونات وبصمة الوقت للملف الأصلي.

نفذنا بعد ذلك الأمر gunzip لفك ضغط الملف. لاحظنا بعدها كيف حلّت النسخة المضغوطة مكان النسخة الأصلية وأيضًا بنفس الأذونات وبصمة الوقت.

هذه بعض خيارات برنامج gzip:

الجدول 18-1: خيارات gzip

الخيار	النتيجة
-c	كتابة المخرجات إلى مجرى الخرج القياسي وإبقاء الملفات الأصلية. يمكن تحديد هذا الخيار أيضًا عن طريق stdout-- و to-stdout--.
-d	فك ضغط. يؤدي هذا الخيار إلى جعل gzip يقوم بنفس عمل gnuzip. يمكن تحديد هذا الخيار أيضًا عن طريق decompress-- أو uncompress--.
-f	إجبار gzip على القيام بعملية ضغط وحتى لو كانت هنالك نسخة من الملف الأصلي. يمكن تحديده أيضًا بواسطة force--.
-h	عرض معلومات الاستخدام. يمكن تحديده باستخدام help--.
-l	عرض إحصائيات الضغط لكل ملف مضغوط. يمكن تحديده أيضًا بواسطة الخيار list--.
-r	الضغط التعاودي أو التكراري (أي الملفات والملفات الموجودة في المجلدات الفرعية وهكذا) إذا كان أحد الوسائط مجلدًا. يمكن تحديد هذا الخيار عن طريق recursive--.
-t	التحقق من محتويات الملف المضغوط. يمكن تحديده أيضًا بواسطة test--.
-v	عرض معلومات عن تقدم عملية الضغط. يمكن تحديده أيضًا عن طريق verbose--.
-number	تحديد مقدار الضغط. number هو رقم يتراوح بين 1 (أسرع لكنه أقل ضغطًا) إلى 9 (أبطئ لكنه أكثر ضغطًا). يمكن التعبير عن القيمتين 1 و 9 بالخيارين fast-- و best-- على التوالي. القيمة الافتراضية هي 6.

لنعد إلى مثالنا السابق:

```
[me@linuxbox ~]$ gzip foo.txt
[me@linuxbox ~]$ gzip -tv foo.txt.gz
```

```
foo.txt.gz: OK
[me@linuxbox ~]$ gzip -d foo.txt.gz
```

لقد استبدلنا في بادئ الأمر الملف foo.txt بالنسخة المضغوطة منه المسماة foo.txt.gz ومن ثم تحققنا من سلامة محتوى الملف المضغوط باستخدام الخيارَين t- و v-. في النهاية، فككنا ضغط الملف وأعدناه إلى شكله الأصلي.

يمكن استخدام gzip استخدامات مثيرة للاهتمام عبر مجريي الدخل والخرج القياسيين:

```
[me@linuxbox ~]$ ls -l /etc | gzip > foo.txt.gz
```

يُنشئ الأمر السابق نسخةً مضغوطةً لقائمة محتويات المجلد etc/.

برنامج gnuzip، الذي يفك ضغط الملفات، يفترض أن امتداد الملف هو ".gz". لذا، ليس من الضروري تحديده لطالما أن اسم الملف المضغوط لا يتعارض مع اسم ملفٍ آخر غير مضغوط:

```
[me@linuxbox ~]$ gunzip foo.txt
```

إذا كان هدفنا فقط هو مشاهدة محتوى الملف، فيمكننا عمل الآتي:

```
[me@linuxbox ~]$ gunzip -c foo.txt | less
```

بشكل بديل، يوجد هنالك برنامج يأتي مع حزمة gzip يُدعى zcat وهو مكافئ لاستخدام برنامج gunzip مع الخيار c-. يُستخدم بشكل مشابه للأمر cat لكن على ملفات gz المضغوطة:

```
[me@linuxbox ~]$ zcat foo.txt.gz | less
```

تلميحة: يوجد هنالك برنامج zless أيضًا. يقوم بنفس عمل الأنبوب السابق.

bzip2

برنامج bzip2 الذي كتبه Julian Seward، هو شبيه ببرنامج gzip لكنه يستخدم خوارزمية ضغط مختلفة تحقق مستويات ضغط أعلى من تلك التي يستخدمها gzip لكنها أبطئ منها. يعمل bzip2 بشكل مشابه جدًا لبرنامج gzip. ينتهي الملف المضغوط باستخدام bzip2 بالامتداد ".bz2":

```
[me@linuxbox ~]$ ls -l /etc > foo.txt
```

```
[me@linuxbox ~]$ ls -l foo.txt
-rw-r--r-- 1 me me 15738 2008-10-17 13:51 foo.txt
[me@linuxbox ~]$ bzip2 foo.txt
[me@linuxbox ~]$ ls -l foo.txt.bz2
-rw-r--r-- 1 me me  2792 2008-10-17 13:51 foo.txt.bz2
[me@linuxbox ~]$ bunzip2 foo.txt.bz2
```

كما لاحظت، يُستخدم برنامج bzip2 بشكل مشابه لبرنامج gzip. جميع الخيارات التي ناقشناها لبرنامج gzip مدعومة في bzip2 (باستثناء r-). لكن لاحظ أن خيار تحديد مقدار الضغط (number-) يكون لديه معنى مختلف بالنسبة إلى bzip2. يأتي bzip2 مع برنامجَيّ bunzip2 و bzcat لفك ضغط الملفات.

يأتي مع حزمة bzip2 البرنامج bzip2recover، الذي يمكن استخدامه لإصلاح ملفات bz2. المتضررة.

لا تسرف بالضغط!

أشاهد العديد من الأشخاص وهم يحاولون ضغط ملف قد ضُغِطَ مُسبقًا باستخدام خوارزمية ضغط فعالة، بتنفيذ أمر شبيه بالآتي:

```
$ gzip picture.jpg
```

لا تفعل ذلك! سوف تنتهي بإضاعة الوقت والحجم التخزيني! إذا ضغطتَ ملفًا مضغوطًا مسبقًا فسينتهي بك الأمر بالحصول على ملفٍ أكبر. هذا لأن تقنيات الضغط تضيف بعض المعلومات التي تصف طريقة الضغط.

أرشفة الملفات

مهمة شائعة لإدارة الملفات بالإضافة إلى الضغط هي الأرشفة. الأرشفة هي عملية جمع العديد من الملفات وتحزيمهم في ملف واحد كبير. يتم القيام بالأرشفة كجزء من عمليات أخذ نسخة احتياطية من الملفات. وتُستخدم أيضًا عند نقل البيانات القديمة من النظام إلى قرص تخزيني طويل الأمد.

tar

الأداة tar هي أداة كلاسيكية لأرشفة الملفات في عالم برمجيات الأنظمة الشبيهة بِيونكس. يشير اسمها (tape archive) إلى جذورها كأداة لإنشاء النسخ الاحتياطية. تشاهد بكثرة ملفات بامتداد tar. أو tgz. التي تشير إلى أرشيفات tar العادية وأرشيفات tar المضغوطة على الترتيب. يتألف أرشيف tar من مجموعة من الملفات، ومجلد أو أكثر. الشكل العام للأمر tar هو:

```
tar mode [options] pathname...
```

حيث mode هو أحد الأنماط الآتية (يحتوي الجدول الآتي على قائمة جزئية، راجع صفحة الدليل للأمر tar للحصول على القائمة الكاملة):

الجدول 18-2: أنماط tar

النمط	الشرح
c	إنشاء أرشيف من قائمة بالملفات و/أو المجلدات.
x	استخراج محتويات الأرشيف.
r	إضافة مسارات محددة إلى نهاية الأرشيف.
t	عرض قائمة بمحتويات الأرشيف.

يستخدم tar طريقة غريبة للتعبير عن الخيارات. لذا، سوف نحتاج إلى بعض الأمثلة لمشاهدة كيف يعمل هذا الأمر. سنُعيد أولًا إنشاء ساحة للتجارب كما في الفصل السابق:

```
[me@linuxbox ~]$ mkdir -p playground/dir-{00{1..9},0{10..99},100}
[me@linuxbox ~]$ touch playground/dir-{00{1..9},0{10..99},100}/file-
{A-Z}
```

ومن ثم سننشئ أرشيف tar بكامل "ساحة التجارب":

```
[me@linuxbox ~]$ tar cf playground.tar playground
```

ينشئ الأمر السابق أرشيف tar باسم playground.tar الذي يحتوي على مجلد playground بكامل محتوياته. يمكننا ملاحظة أن النمط المستخدم والخيار f، الذي يُستخدم لتحديد اسم أرشيف tar الناتج، يمكن أن يدمجا سويةً ولا يحتاجان إلى شرطة قبلهما. لكن لاحظ أن النمط يُحدَّد دائمًا قبل أيّة خيارات.

نستخدم الأمر الآتي لعرض قائمة بمحتويات ملف tar:

```
[me@linuxbox ~]$ tar tf playground.tar
```

نضيف الخيار v (verbose) لنحصل على قائمة أكثر تفصيلًا:

```
[me@linuxbox ~]$ tar tvf playground.tar
```

لنستخرج الآن محتويات playground.tar في مكانٍ جديد. سنقوم بذلك بإنشاء مجلد جديد باسم foo

وتغيير مجلد العمل الحالي واستخراج محتويات أرشيف tar:

```
[me@linuxbox ~]$ mkdir foo
[me@linuxbox ~]$ cd foo
[me@linuxbox foo]$ tar xf ../playground.tar
[me@linuxbox ~]$ ls
playground
```

إذا تفحصنا محتويات foo/playground/~، فسوف نلاحظ أن الملف أُستخرج استخراجًا صحيحًا. لكن هنالك إشكالية صغيرة هي أن مالك الملفات المُستخرَجة هو المستخدم الذي قام بالاستخراج وليس المالك الأصلي، إلا إذا كنت تقوم بالاستخراج بحساب الجذر.

سلوك آخر مثير للاهتمام لبرنامج tar هو طريقة تعامله مع مسارات الملفات. نوع المسارات الافتراضي هو المسارات النسبية وليس المطلقة. يقوم tar بذلك بإزالة الخط المائل من بداية المسار عند إنشاء الأرشيف. لإزالة الغموض، سنعيد إنشاء الأرشيف السابق لكن مع تحديد مسار مطلق عوضًا عن مسار نسبي:

```
[me@linuxbox foo]$ cd
[me@linuxbox ~]$ tar cf playground2.tar ~/playground
```

تذكر أن playground/~ سيُوسّع إلى home/me/playground/. سنستخرج الآن الملف، وسنحصل في النهاية على مسار مطلق:

```
[me@linuxbox ~]$ cd foo
[me@linuxbox foo]$ tar xf ../playground2.tar
[me@linuxbox foo]$ ls
home  playground
[me@linuxbox foo]$ ls home
me
[me@linuxbox foo]$ ls home/me
playground
```

بعد استخراجنا للأرشيف الجديد، أُنشِئ مجلد home/me/playground في مجلد العمل الحالي foo/~ وليس في المجلد الجذر "/". قد يبدو هذا السلوك غريبًا، لكنه مفيد أكثر حيث يسمح لنا باستخراج محتويات الأرشيف إلى أي مجلد بدلًا من إجبارنا على استخراجه لمجلد محدد مسبقًا. ستُعطيك إعادة التمرين باستخدام الخيار "v" صورة أوضح عما يجري.

لنفترض على سبيل المثال أننا نريد نسخ مجلد المنزل بكل محتوياته من نظامنا إلى نظامٍ آخر وسنستخدم

قرص USB ذا حجمٍ تخزينيٍ كبير لنقل الأرشيف. يُوصَل قرص USB تلقائيًّا في مجلد media/ في توزيعات لينُكس الحديثة. ولنفترض أن اسم قرص USB هو BigDisk. لذا، ننفذ الأمر الآتي لإنشاء الأرشيف:

```
[me@linuxbox ~]$ sudo tar cf /media/BigDisk/home.tar /home
```

بعد كتابة الملف على القرص، سنفصله ومن ثم ندرجه في الحاسوب الثاني. ننفذ الأمر الآتي لكي نستخرج محتويات الأرشيف:

```
[me@linuxbox2 ~]$ cd /
[me@linuxbox2 /]$ sudo tar xf /media/BigDisk/home.tar
```

المهم هنا هو ملاحظة أننا غيرنا مجلد العمل الحالي إلى "/" أولًا، لكي تُستخرج محتويات الأرشيف بالنسبة إلى المجلد الجذر؛ لأن جميع المسارات في الأرشيف نسبية.

عندما نستخرج محتويات أرشيفٍ ما، فمن الممكن أن نُحدِّد ما الذي نريد استخراجه. على سبيل المثال، إذا أردنا استخراج ملف واحد فقط من الأرشيف، فإننا نكتب الأمر كالآتي:

```
tar xf archive.tar pathname
```

بإضافة الوسيط pathname إلى الأمر، فسيتخرج tar الملف المحدد فقط. يمكن تحديد أكثر من ملف لكي يتم استخراجهم. لاحظ أن المسار يجب أن يكون كاملًا كما هو مخزن في الأرشيف. لا يُسمح باستخدام المحارف البديلة عند تحديد المسارات؛ لكن نسخة غنو من tar (وهي النسخة التي تتوفر في الغالبية العظمى من توزيعات لينُكس) تدعم الخيار wildcards-- . هذا مثالٌ عنها:

```
[me@linuxbox ~]$ cd foo
[me@linuxbox foo]$ tar xf ../playground2.tar --wildcards
'home/me/playground/dir-*/file-A'
```

سيستخرج الأمر السابق جميع الملفات التي تطابق المسار المحدد الذي يحتوي على نمط المحارف الخاصة dir-*.

يُستخدم tar عادةً مع find لإنشاء الأرشيفات. سنستخدم find، في هذا المثال، للعثور على الملفات ومن ثم إضافتها إلى الأرشيف:

```
[me@linuxbox ~]$ find playground -name 'file-A' -exec tar rf play-
ground.tar '{}' '+'
```

استخدمنا الأمر find للعثور على جميع الملفات الموجودة في مجلد playground التي تحمل الاسم

file-A ومن ثم جعلنا tar (باستخدام الفعل exec-) يضيفها لأرشيف playground.tar باستخدام نمط الإضافة "r".

استخدام tar مع find هو طريقة جيدة لإنشاء "نِسخ احتياطية تراكميّة" (incremental backups) لمجلد معين أو لجميع النظام. باستخدام find للعثور على الملفات الأحدث من ملف مرجعي (على فرض أن الملف المرجعي يُحدَّث بعد إنشاء الأرشيف مباشرةً)، يمكننا إنشاء أرشيف جديد يحتوي على الملفات التي عُدِّلت فقط بالنسبة إلى الأرشيف القديم.

يمكن أن يُستخدم tar مع مجريي الدخل والخرج القياسيَين. يلخص المثال الآتي هذه الفكرة:

```
[me@linuxbox foo]$ cd
[me@linuxbox ~]$ find playground -name 'file-A' | tar cf - --files-
from=- | gzip > playground.tgz
```

استخدمنا في المثال السابق برنامج find لتوليد قائمة بالملفات المطابقة للبحث ومن ثم مررناها عبر الأنبوب إلى tar، إذا كان اسم الملف هو "-" فهذا يعني أن اسم الملف سيأتي من مجرى الدخل (بالمناسبة، يُعتَمَد استخدام "-" في أسماء الملفات للدلالة على مجريي الدخل أو الخرج القياسيَين من قِبل العديد من البرامج). الخيار files-from-- (يمكن تحديده بالخيار T-) يؤدي إلى جعل tar يقرأ أسماء الملفات من ملف وليس عبر تحديدهم كوسائط. أخيرًا، مُرِّرَ الأرشيف الناتج عبر الأنبوب إلى الأمر gzip لإنشاء أرشيف مضغوط playground.tgz. الامتداد tgz. هو امتداد شهير يُستخدم للإشارة إلى أرشيفات tar المضغوطة باستخدام gzip. يُستخدم في بعض الأحيان الامتداد "tar.gz.".

وعلى الرغم من أننا استخدمنا برنامج gzip لإنشاء الأرشيف المضغوط؛ إلا أن الإصدارات الحديثة من غنو tar تدعم الضغط باستخدام gzip او bzip2 مباشرةً، وذلك باستخدام الخيارَين "z" و "j" على التوالي. يمكننا تبسيط المثال السابق لكي يصبح كالآتي:

```
[me@linuxbox ~]$ find playground -name 'file-A' | tar czf play-
ground.tgz -T -
```

إذا أردنا إنشاء أرشيف مضغوط باستخدام bzip2، فإننا نعدل الأمر السابق إلى:

```
[me@linuxbox ~]$ find playground -name 'file-A' | tar cjf play-
ground.tbz -T -
```

بكل بساطة عن طريق تعديل خيار الضغط من z إلى j (وتغيير امتداد الملف الناتج إلى tbz.) فإننا نُفَعِّل الضغط باستخدام bzip2.

استخدام آخر مثير للاهتمام لمجريي الدخل والخرج في tar هو نقل الملفات بين الأنظمة عبر الشبكة. تخيل أن لدينا جهازين يعمل كلاهما بنظام شبيه بِيونكس يوفر الأمرَين tar و ssh. يمكننا نقل مجلد ما من النظام البعيد (يسمى remote-sys على سبيل المثال) إلى نظامنا المحلي:

```
[me@linuxbox ~]$ mkdir remote-stuff
[me@linuxbox ~]$ cd remote-stuff
[me@linuxbox remote-stuff]$ ssh remote-sys 'tar cf - Documents' | tar
xf -
me@remote-sys’s password:
[me@linuxbox remote-stuff]$ ls
Documents
```

تمكّنا في المثال السابق من نقل مجلد يسمى Documents من النظام البعيد إلى مجلد موجود داخل مجلد آخر مسمى remote-stuff في النظام المحلي. لكن كيف تم ذلك؟ أولًا، شغلنا برنامج tar في النظام البعيد عن طريق ssh. تتذكر أن ssh يسمح لنا بتنفيذ الأوامر على النظام البعيد و "مشاهدة" ناتجها على النظام المحلي، الخرج القياسي الذي أُنشئ في النظام البعيد أُرسل إلى النظام المحلي. يمكننا الاستفادة من هذا بجعل البرنامج tar ينشئ أرشيفًا (بالنمط c) ويرسل الناتج إلى مجرى الخرج القياسي بدلًا من ملف (استخدام الخيار f مع الشرطة). وبعد نقل الأرشيف عبر النفق المشفر إلى الجهاز المحلي؛ نفذنا الأمر tar واستخرجنا محتويات الأرشيف (النمط x).

zip

البرنامج zip هو أداة ضغط وأرشفة في آن واحد. تكون صيغة الملف الناتج مألوفة لمستخدمي ويندوز حيث يستطيع ويندوز قراءة وكتابة ملفات zip. لكن في لينُكس، تُضغط الملفات بشكل رئيسي باستخدام gzip ويتبعه bzip2 كخيار ثانوي.

كأبسط أشكال الاستخدام، يُستخدم zip بالطريقة الآتية:

```
zip options zipfile file...
```

لإنشاء ملف zip مضغوط لمجلد playground، ننفذ الأمر الآتي:

```
[me@linuxbox ~]$ zip -r playground.zip playground
```

إذا لم نستخدم الخيار ‎-r، فسيُضاف المجلد فقط دون إضافة أيٍّ من محتوياته. وعلى الرغم من أن إضافة الامتداد ‎.zip لاسم الملف الناتج هو أمر غير ضروري، إلا أننا قمنا بذلك للتوضيح.

يعرض برنامج zip سلسلة من الرسائل شبيهة بالرسائل الآتية أثناء إنشاء أرشيف zip:

```
adding: playground/dir-020/file-Z (stored 0%)
adding: playground/dir-020/file-Y (stored 0%)
adding: playground/dir-020/file-X (stored 0%)
adding: playground/dir-087/ (stored 0%)
adding: playground/dir-087/file-S (stored 0%)
```

تُظهر هذه الرسائل حالة كل ملف يضاف إلى الأرشيف. يضيف برنامج zip الملفات إلى الأرشيف بطريقتين: إما أن "يخزنها" كما هي دون أيّ ضغط، أو أن يضيفها بعد الضغط. القيمة الرقمية التي تظهر في آخر الرسالة تُظهر مقدار الضغط الذي طُبِّق على الملف. ولأن جميع الملفات المُضافة هي ملفات فارغة، فلن يتم إجراء الضغط عليها.

استخراج محتويات ملف zip مضغوط هو أمر سهل جدًا وذلك باستخدام البرنامج unzip:

```
[me@linuxbox ~]$ cd foo
me@linuxbox:~/foo$ unzip ../playground.zip
```

يجب الانتباه إلى سلوك في zip (وهو عكس السلوك في tar) أنه إذا حُدِّدَ أرشيف موجود مسبقًا فلن يُستبدل بل ستُضاف الملفات إليه. هذا يعني أن الأرشيف سيبقى، لكن الملفات الجديدة ستُضاف إليه وتُستبدل الملفات الموجودة مسبقًا في حال إضافة ملف بنفس اسمها إلى الأرشيف.

يمكن عرض واستخراج الملفات من أرشيف zip انتقائيًا وذلك بتمرير مساراتهم إلى الأمر unzip:

```
[me@linuxbox ~]$ unzip -l playground.zip playground/dir-087/file-Z
Archive: ../playground.zip
  Length        Date      Time        Name
 --------       ----      ----        ----
        0     10-05-08   09:25        playground/dir-087/file-Z
 --------                             -------
        0                             1 file
[me@linuxbox ~]$ cd foo
me@linuxbox:~/foo$ unzip ./playground.zip playground/dir-087/file-Z
Archive: ../playground.zip
replace playground/dir-087/file-Z? [y]es, [n]o, [A]ll, [N]one,
[r]ename: y
 extracting: playground/dir-087/file-Z
```

استخدام الخيار l- يجعل برنامج unzip يُظهر قائمة بمحتويات الأرشيف دون استخراج أيّ ملف. إذا لم يُحدَّد ملف/ملفات كوسيط، فسيعرض unzip جميع الملفات في الأرشيف. يُستخدم الخيار v- لزيادة المعلومات التي ستُعرَض. لاحظ أن المستخدم يُسأل فيما إذا كان يريد استبدال الملف إذا تضاربت عملية استخراج الملفات من الأرشيف مع الملفات الموجودة مسبقًا.

وكما في برنامج tar، يسمح zip باستخدام مجريي الدخل والخرج القياسيَين. يمكن أن تُمرِّر أسماء الملفات عبر الأنبوب إلى الأمر zip باستخدام الخيار "@-":

```
me@linuxbox:~/foo$ cd
[me@linuxbox ~]$ find playground -name "file-A" | zip -@ file-A.zip
```

استخدمنا هنا الأمر find لتوليد قائمة بالملفات التي تُطابِق الاختبار "name "file-A- ومررنا الناتج إلى zip، الذي بدوره يُنشئ الأرشيف file-A.zip الذي يحتوي على الملفات المُحددة.

يسمح zip أيضًا بالكتابة إلى مجرى الخرج القياسي، لكن استخدامه محدود لأن عددًا قليلًا جدًا من البرامج تستفيد من الناتج. لسوء الحظ، لا يقبل برنامج unzip المدخلات من مجرى الدخل القياسي. وهذا ما يمنع استخدام برنامجي zip و unzip بغرض نقل الملفات عبر الشبكة كما في البرنامج tar.

لكن برنامج zip يقبل المدخلات من مجرى الدخل القياسي، لذا، فإننا نستطيع استخدامه لضغط ناتج البرامج الأُخرى:

```
[me@linuxbox ~]$ ls -l /etc/ | zip ls-etc.zip -
adding: - (deflated 80%)
```

مررنا، في المثال السابق، ناتج الأمر ls إلى zip كي يضغطه. وكما في tar، يستعمل zip الشرطة لكي يستخدم مجرى الدخل القياسي عوضًا عن الملفات.

يسمح برنامج unzip بإرسال مخرجاته إلى مجرى الخرج القياسي باستخدام الخيار p- (اختصار للكلمة pipe):

```
[me@linuxbox ~]$ unzip -p ls-etc.zip | less
```

لقد شرحنا الكثير من أساسيات zip و unzip. يملك كلاهما العديد من الخيارات لإضافة الكثير من المرونة لهما. صفحة الدليل man للأمرين zip و unzip زاخرة بالمعلومات وحتى الأمثلة! لكن الغرض الأساسي من هذين البرنامجَين هو السماح بتبادل الملفات مع نظام ويندوز، وليس القيام بعمليات الضغط في لينُكس؛ حيث يفضل في تلك الحالة استخدام tar و gzip.

مزامنة الملفات والمجلدات

استراتيجية شهيرة تستخدم لإدارة النسخ الاحتياطية للنظام هي إبقاء مجلد أو أكثر متزامنًا مع مجلد أو أكثر موجود في جهاز تخزين آخر (يكون غالبًا جهاز تخزين قابل للإزالة) في نفس النظام أو في نظام بعيد. يمكننا على سبيل المثال، أن نزامن نسخة من موقع وِب قيد التطوير من وقتٍ إلى آخر مع خادم الوِب الأساسي.

الأداة المفضلة لهذه المهمة في عالم الأنظمة الشبيهة بِيونكس هي rsync. يزامن هذا البرنامج المجلدين المحلي والبعيد باستخدام بروتوكول يسمى remote-update، الذي يسمح لبرنامج rsync بشكل سريع بمعرفة الاختلافات بين المجلدين والقيام بأقل قدر من النسخ لمزامنتهما. وهذا ما يجعل rsync سريعًا للغاية مقارنةً مع باقي برامج النسخ.

يُنَفّذ برنامج rsync كالآتي:

rsync *options source destination*

حيث المصدر (source) أو الوِجهة (destination) هو واحدٌ من ما يلي:

- ملف أو مجلد محلي.
- ملف أو مجلد "بعيد" على الشكل [user@]host:path.
- خادم rsync بعيد يُحدَّد بالشكل rsync://[user@]host[:port]/path.

انتبه إلى أن المصدر أو الوجهة يجب أن يكون ملفًا محليًا. النسخ ما بين نظامين بعدين ليس أمرًا مدعومًا في rsync.

لنجرب rsync على بعض الملفات المحلية. لنحذف بادئ الأمر محتويات مجلد foo:

```
[me@linuxbox ~]$ rm -rf foo/*
```

لنزامن الآن المجلد playground بإنشاء نسخة مطابقة له في foo:

```
[me@linuxbox ~]$ rsync -av playground foo
```

استخدمنا الخيارين a- (الأرشفة. للمحافظة على خصائص الملفات) وv- (verbose) لإنشاء نسخة "معكوسة" (mirror) لمجلد playground في foo. سنشاهد قائمةً بالملفات التي تُنسَخ في أثناء تنفيذ الأمر rsync. في النهاية، سنشاهد رسالة كالرسالة الآتية:

```
sent 135759 bytes received 57870 bytes  387258.00 bytes/sec
total size is 3230 speedup is 0.02
```

التي تُظهر مقدار البيانات التي نُسخت. إذا نفذنا الأمر مرّةً ثانيةً، فسوف تظهر لنا نتيجة مختلفة:

```
[me@linuxbox ~]$ rsync -av playgound foo
building file list ... done

 sent 22635 bytes received 20 bytes  45310.00 bytes/sec
total size is 3230 speedup is 0.14
```

لاحظ عدم وجود قائمة بالملفات. هذا لأن rsync عَرَفَ أنه لا توجد أيّة اختلافات بين ‎~/playgound‎ و ‎~/foo/playgound‎. لذا، لن يحتاج إلى نسخ أي شيء. إذا عدّلنا في مجلد playgound ومن ثم نفذنا الأمر rsync مجددًا:

```
[me@linuxbox ~]$ touch playground/dir-099/file-Z
[me@linuxbox ~]$ rsync -av playground foo
building file list ... done
playground/dir-099/file-Z
sent 22685 bytes received 42 bytes 45454.00 bytes/sec
total size is 3230 speedup is 0.14
```

كما لاحظت، قد وجد rsync اختلافًا ونسخ الملف الجديد فقط.

كتطبيقٍ عملي، سنستخدم قرص USB الذي استخدمناه سابقًا مع أمثلة tar. إذا أدرجنا القرص في الجهاز ووُصِلَ إلى المجلد ‎/media/BigDisk‎، فسنستطيع إنشاء نسخة احتياطية بإنشاء مجلد باسم backup في قرص USB ومن ثم ننسخ أهم الأشياء الموجودة على جهازنا باستخدام rsync:

```
[me@linuxbox ~]$ mkdir /media/BigDisk/backup
[me@linuxbox ~]$ sudo rsync -av --delete /etc /home /usr/local
/media/BigDisk/backup
```

نسخنا، في المثال السابق، المجلدات ‎/etc‎ و ‎/home‎ و ‎/usr/local‎ من نظامنا إلى قرص USB. استخدمنا الخيار ‎--delete‎ لحذف الملفات الموجودة في مجلد backup في قرص USB التي لم تعد موجودةً في النظام المحلي (قد لا يفيد هذا الخيار كثيرًا في أول مرّة نأخذ فيها النسخة الاحتياطية؛ لكنه مفيد بعد ذلك). تكرار عملية إدراج القرص الذي يحتوي على النسخة الاحتياطية وتنفيذ الأمر rsync قد تكون طريقة مفيدة لإجراء النسخ الاحتياطي للنظام. يمكننا إنشاء أمر بديل لتجنب كتابة كامل الأمر عند القيام بالنسخ الاحتياطي، ثم نضيفه إلى ملف ".bashrc":

```
alias backup='sudo rsync -av --delete /etc /home /usr/local
/media/BigDisk/backup'
```

كل ما نحتاج إلى فعله هنا هو إدراج قرص USB وتنفيذ الأمر backup.

استخدام الأمر rsync عبر الشبكة

إحدى أهم ميزات rsync هي إمكانية استخدامه لنسخ الملفات عبر الشبكة. الحرف "r" في rsync يرمز لكلمة "remote" أي بعيد. النسخ الاحتياطي عن بعد يمكن أن يتم بإحدى الطريقتين الآتيتين: الطريقة الأولى تتطلب خادم يحتوي على rsync بالإضافة إلى ssh. لنفترض أن لدينا حاسوبًا آخر موصولًا بالشبكة ويحتوي على الكثير من المساحة التخزينية الفارغة ونريد أن نأخذ النسخة الاحتياطية على ذاك النظام بدلًا من قرص USB. بفرض أن النظام الآخر يحتوي على مجلد باسم backup، فنستطيع تنفيذ الأمر الآتي:

```
[me@linuxbox ~]$ sudo rsync -av --delete --rsh=ssh /etc /home
/usr/local remote-sys:/backup
```

لقد قمنا بتغييرين على الأمر حتى نستطيع النسخ عبر الشبكة. أولًا أضفنا الخيار rsh=ssh--. الذي يجعل rsync يستخدم ssh، وبالتالي نستطيع نقل الملفات عبر نفق آمن بين الحاسوب الحالي والحاسوب البعيد. ثانيًا، حددنا المضيف البعيد بكتابة اسمه قبل المسار (اسم النظام البعيد في هذه الحالة هو remote-sys).

الطريقة الثانية التي يستخدم فيها rsync لمزامنة الملفات عبر الشبكة هي استخدام خادم rsync. يمكن إعداد rsync للعمل كخادم و "الاستماع" إلى طلبات التزامن. يتم ذلك عادةً للسماح بإنشاء نسخة انعكاسية (mirroring) من النظام البعيد. على سبيل المثال، تطرح ريدهات مجموعةً كبيرةً من البرمجيات التي تكون قيد التطوير لتوزيعة فيدورا. من المفيد لمجربي البرمجيات أن ينسخوا هذه المجموعة في أثناء فترة تطوير التوزيعة للتبليغ عن العِلل. ولما كانت الملفات في المستودع تتغير تغيرًا مستمرًا (أكثر من مرّة كل يوم)، فمن المحبذ أن تتم متزامنة النظام المحلي مع النظام البعيد بدل نسخ كل ملفات المستودع. أحد تلك المستودعات موجود في Georgia Tech؛ يمكننا أن ننسخه على جهازنا كالآتي:

```
[me@linuxbox ~]$ mkdir fedora-devel
[me@linuxbox ~]$ rsync -av -delete rsync://rsync.gtlib.gatech.edu/fed
ora-linux-core/development/i386/os fedora-devel
```

استخدمنا في الأمر السابق رابط خادم rsync البعيد، الذي يتكون من البروتوكول (//:rsync) ويتبعه اسم المضيف (rsync.gtlib.gatech.edu) ومن ثم مسار المستودع.

الخلاصة

لقد ألقينا نظرة على برامج الضغط والأرشفة المُستخدمة في لينُكس وباقي الأنظمة الشبيهة بِيونكس. يُفضَّل استخدام tar/gzip عند أرشفة الملفات في الأنظمة الشبيهة بِيونكس. بينما يُستخدم zip/unzip للسماح بتبادل الأرشيف مع أنظمة ويندوز. في النهاية، ألقينا نظرة على برنامج rsync (أحد برامجي المُفضّلة) الذي يفيد للغاية في مزامنة الملفات والمجلدات بين الأنظمة.

الفصل التاسع عشر:

التعابير النظامية

سنلقي نظرة في الفصول القليلة القادمة على بعض الأدوات التي تُستخدم لمعالجة النصوص. وكما رأينا سابقًا، تلعب البيانات النصية دورًا مهمًا في جميع الأنظمة الشبيهة بِيونكس بما فيها لينُكس. لكن قُبيل البدء في شرح جميع ميزات تلك الأدوات، يجب علينا تعلم تقنية تستخدم بكثر مع أغلب الاستخدامات المعقدة لتلك الأدوات، ألا وهي التعابير النظامية (Regular Expressions).

بينما كنا نتعلم المزايا المتعددة التي يوفرها سطر الأوامر، واجهنا العديد من الميزات والأوامر؛ كالتوسعات، والاقتباسات، واختصارات لوحة المفاتيح، وتأريخ الأوامر، ولا ننسَ محرر `vi`. تُكمل التعابير النظامية هذه المسيرة "التقليدية" ويمكن تصنيفها (أي التعابير النظامية) على أنها أهم ميزة على الإطلاق! الفهم الجيد للتعابير النظامية يُمكّننا من القيام بالعديد من المهمات المعقدة والصعبة؛ لكن قد لا يظهر أثر تعلمها مباشرةً في هذا الفصل.

ما هي التعابير النظامية؟

ببساطة، يمكن تعريف التعابير النظامية على أنها مجموعة من المحارف (حروف ورموز) تُستخدم لتمثيل أنماط نصية. إنها تشابه ميزة المحارف البديلة التي توفرها الصدفة لمطابقة أسماء الملفات، إلا أنها أوسع وأشمل. تُدعَم التعابير النظامية من قِبل العديد من أدوات سطر الأوامر وقد أُعتمِدَت في لغات برمجية كثيرة للقيام بعمليات معالجة النصوص. لكن قد تصبح الأمور مُربِكةً بعض الشيء عند معرفة أنه ليست جميع التعابير النظامية متماثلة؛ وهي تختلف من أداة لأخرى ومن لغة برمجة إلى أخرى. سنجعل نقاشنا مُقتصِرًا على التعابير التي يدعمها معيار POSIX (الذي تعتمده أغلب الأدوات في سطر الأوامر)، وعلى النقيض من أغلب لغات البرمجة (وأشهرها في هذا المجال هي بيرل) التي تدعم مجموعة أكبر وأشمل من الرموز والمحارف.

grep

البرنامج الأساسي الذي سنستخدمه للتعامل مع التعابير النظامية هو صديقنا القديم `grep`. الكلمة "`grep`" في الواقع مُستقاة من الجملة "global regular expression print". لذا، يمكننا ملاحظة أن `grep` مرتبط بالتعابير النظامية بشكلٍ أو بآخر. يبحث `grep` في الملفات النصية عن مطابقات لتعبير نظامي مُحَدَّد ويطبع أي سطر يحتوي على ذاك النمط إلى مجرى الخرج القياسي.

لقد استخدمنا grep، حتى الآن، لمطابقة النصوص البسيطة "الثابتة" كالآتي:

```
[me@linuxbox ~]$ ls /usr/bin | grep zip
```

يطبع الأمر السابق جميع الملفات الموجودة في مجلد usr/bin/ التي يحتوي اسمها على العبارة "zip".

يقبل برنامج grep الخيارات والوسائط على النحو الآتي:

grep [*options*] regex [*file*...]

حيث regex هو نمط التعابير النظامية.

يحتوي الجدول الآتي على قائمة بأشهر خيارات البرنامج grep:

الجدول 19-1: خيارات grep

الخيار	الشرح
-i	تجاهل حالة الأحرف. عدم التفريق ما بين الأحرف الكبيرة (uppercase) والأحرف الصغيرة (lowercase). يمكن تحديده أيضًا عن طريق الخيار ignore-case--.
-v	عكس التحديد. افتراضيًا، يطبع grep الأسطر التي تحتوي على مطابقة أو أكثر. هذا الخيار يجعل grep يطبع كل سطر لا يحتوي على أيّة مطابقات. يمكن تحديده عن طريق الخيار invert-match--.
-c	طباعة عدد المطابقات (أو عدد الأسطر غير المطابقة إن أُستخدم الخيار v-) عوضًا عن الأسطر أنفسهم. يمكن تحديده أيضًا بالخيار count--.
-l	طباعة اسم كل ملف يحتوي على المطابقة بدلًا من طباعة الأسطر أنفسهم. يمكن تحديده أيضًا بالخيار files-with-matches--.
-L	شبيه بالخيار l-؛ لكنه يجعل grep يطبع أسماء الملفات التي لا تحتوي على أيّة مطابقات. يمكن تحديده أيضًا بالخيار files-without-match--.
-n	إسباق كل سطر مُطابَق برقم ذاك السطر في الملف الأصلي. يمكن تحديده أيضًا بالخيار line-number--.
-h	عدم طباعة اسم الملف عند إجراء بحث في أكثر من ملف. يمكن تحديده أيضًا بالخيار no-filename--.

سنُنشئ بعض الملفات النصية التي سنبحث في محتواها؛ لكي نستكشف grep استكشافًا كاملًا:

```
[me@linuxbox ~]$ ls /bin > dirlist-bin.txt
[me@linuxbox ~]$ ls /usr/bin > dirlist-usr-bin.txt
[me@linuxbox ~]$ ls /sbin > dirlist-sbin.txt
[me@linuxbox ~]$ ls /usr/sbin > dirlist-usr-sbin.txt
[me@linuxbox ~]$ ls dirlist*.txt
dirlist-bin.txt        dirlist-sbin.txt           dirlist-usr-sbin.txt
dirlist-usr-bin.txt
```

نستطيع القيام ببحث صغير في قائمة الملفات كالآتي:

```
[me@linuxbox ~]$ grep bzip dirlist*.txt
dirlist-bin.txt:bzip2
dirlist-bin.txt:bzip2recover
```

بحث grep، في المثال السابق، في كل الملفات الموجودة عن السلسلة النصية "bzip" ووجد مطابقتَين كلاهما في الملف dirlist-bin.txt. إذا كنت مهتمًا بأسماء الملفات التي تحتوي على المطابقات وليس المطابقات أنفسهم، فتستطيع استخدام الخيار "l-":

```
[me@linuxbox ~]$ grep -l bzip dirlist*.txt
dirlist-bin.txt
```

بشكل مشابه، إذا أردنا مشاهدة قائمة بالملفات التي لا تحتوي على مطابقات، فإننا ننفذ الأمر الآتي:

```
[me@linuxbox ~]$ grep -L bzip dirlist*.txt
dirlist-sbin.txt
dirlist-usr-bin.txt
dirlist-usr-sbin.txt
```

الحروف العادية والرموز الخاصة

ربما لم يظهر لك جليًا أن جميع عمليات البحث التي قمت باستخدام grep فيها حتى الآن تستخدم التعابير النظامية بشكلٍ أو بآخر، بما فيها أبسط عمليات البحث. التعبير النظامي "bzip" يعني أن المطابقة تحدث فقط إذا حوى سطرٌ ما في ملف أربعة محارف على الأقل وكان في ذلك السطر الأحرف "b" و"z "و"i" و"p" بالترتيب ذاته ودون وجود أيّة محارف تفصل بينهما. المحارف التي تتكون منها الكلمة "bzip" يُطلق عليها اصطلاحيًا "الحروف العادية" (literal characters)، أي تلك المحارف التي تُطابِق نفسها فقط. تحتوي التعابير

النظامية بالإضافة إلى الأحرف العادية ما يُسمى "الرموز الخاصة" (meta-characters) التي تُستخدم لتحديد مطابقات أكثر تعقيدًا. تتكون الرموز الخاصة التي تحتوي عليها التعابير النظامية من الرموز الآتية:

```
^ $ . [ ] { } - ? * + ( ) | \
```

تعتبر جميع المحارف الأخرى أحرفًا عادية. يجدر بالذكر بأن الشرطة المائلة الخلفية "\" تُستخدم في بضع حالاتٍ لإنشاء سلاسل خاصة (meta sequences) وأيضًا للسماح للرموز الخاصة بأن تُهرَّب وتُعامَل كأحرف عادية.

ملاحظة: كما رأينا في الفقرة السابق، إن عددًا من الرموز الخاصة المُستخدمة في التعابير النظامية يكون لها معنى خاص بالنسبة إلى الصدفة عندما تتم عملية التوسعة. عندما نُمرِّر نمط تعابير نظامية يحتوي على أحد الرموز الخاصة إلى سطر الأوامر، فيكون من المهم جدًا أن نضع النمط ما بين علامتي اقتباس (لمنع الصدفة من توسعتهم).

محرف الـ"أي شيء"

أول رمز من الرموز الخاصة الذي سنناقشه هو رمز النقطة "." الذي يُستخدم لمطابقة أي محرف. إذا وضعناه في تعبير نظامي، فإنه سيطابق أي محرف في ذاك الموضع. مثال:

```
[me@linuxbox ~]$ grep -h '.zip' dirlist*.txt
bunzip2
bzip2
bzip2recover
gunzip
gzip
funzip
gpg-zip
preunzip
prezip
prezip-bin
unzip
unzipsfx
```

لقد بحثنا في ملفاتنا عن أي سطر يُطابِق التعبير النظامي "`.zip`". هناك شيئان مثيران للاهتمام في الناتج. لاحظ أن البرنامج `zip` لم يُضمَّن في القائمة. هذا لأن رمز النقطة في التعبير النظامي زاد عدد الأحرف المطلوبة إلى أربعة حروف، ولأن الاسم "`zip`" لا يحتوي إلا على ثلاثة حروف، فإنه لم يُطابَق. أيضًا إذا حوت

القائمة على الامتداد "zip." فسيطابق أيضًا؛ لأن رمز النقطة في امتداد الملف يُعامَل كحرف عادي.

بداية ونهاية السطر

يرمز المحرفان "^" و "$" في التعابير النظامية إلى بداية ونهاية السطر على التوالي. هذا يعني أنهما يؤديان إلى المطابقة في حال وُجِدَ نمط التعابير النظامية في بداية السطر ("^") أو نهاية السطر ("$"):

```
[me@linuxbox ~]$ grep -h '^zip' dirlist*.txt
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
[me@linuxbox ~]$ grep -h 'zip$' dirlist*.txt
gunzip
gzip
funzip
gpg-zip
preunzip
prezip
unzip
zip
[me@linuxbox ~]$ grep -h '^zip$' dirlist*.txt
zip
```

بحثنا في المثال المثال السابق عن السلسلة النصية "zip" الموجودة عند بداية السطر، ثم عند آخر السطر، ثم الموجودة عند بداية ونهاية السطر (أي أن الكلمة "zip" موجودة في سطرٍ بأكمله). لاحظ أن النمط "^$" (بداية السطر ونهايته ولا شيء بينهما) سيُطابِق الأسطر الفارغة.

مساعد حلّ الكلمات المتقاطعة

وحتى بمعرفتنا المتواضعة بالتعابير النظامية نستطيع أن نقوم بشيء مفيد!

يحب أخي الصغير حلّ الكلمات المتقاطعة ويطلب مني أحيانًا المساعدة في سؤالٍ معين يشبه السؤال

الآتي: "ما هي الكلمة التي بطول خمسة حروف التي يكون الحرف الثالث فيها هو 'j' والحرف الأخير 'r' وتعني...؟" قد يتطلب مثل هذا السؤال تفكيرًا كثيرًا.

هل تعلم أن نظام لينُكس يحتوي على قاموس؟ ألقِ نظرة في المجلد /usr/share/dict وستجد واحدًا أو أكثر. ملفات القاموس الموجودة هناك هي مجرد ملفات تحتوي على قائمةٍ طويلةٍ بالكلمات، كل كلمة بسطر، مرتبةً ترتيبًا هجائيًّا. في النظام الخاص بي، يحتوي ملف words على أكثر من 98500 كلمة. للحصول على جواب سؤال الكلمات المتقاطعة السابق، فإننا ننفذ الأمر الآتي:

```
[me@linuxbox ~]$ grep -i '^..j.r$' /usr/share/dict/words
Major
major
```

باستخدام هذا التعبير، استطعنا معرفة جميع الكلمات الموجود في القاموس التي طولها هو خمسة حروف ويكون الحرف "j" هو الحرف الثالث والحرف "r" هو الحرف الأخير.

تعابير الأقواس وفئات الأحرف

بالإضافة إلى مطابقة أي محرف في مكان معين في نمط التعابير النظامية، يمكننا أيضًا مطابقة حرف واحد لأحد عناصر مجموعة محددة من الحروف باستخدام تعابير الأقواس (bracket expressions). يمكننا تحديد مجموعة من المحارف باستخدام تعابير الأقواس (بما فيها المحارف التي تعتبر من الأحرف الخاصة) التي ستُطابق. في هذا المثال، سنستخدم مجموعة تحتوي على حرفين فقط:

```
[me@linuxbox ~]$ grep -h '[bg]zip' dirlist*.txt
bzip2
bzip2recover
gzip
```

طابقنا أي سطر يحتوي على إحدى الكلمتين "bzip" أو "gzip".

يمكن للمجموعة أن تحتوي على أي عدد من المحارف، وستفقد فيها الأحرف الخاصة معناها وتعامل على أنها أحرف عادية. لكن يوجد هنالك حرفَين خاصَين لا يفقدان معناهما هما: "^" الذي يعني "رفض" (أو أخذ معاكس) المجموعة الحالية؛ والشرطة "-" التي تُستخدم لتحديد مجال المحارف كما سنرى لاحقًا.

الرفض

إذا كان أول محرف في تعبير الأقواس هو "^"، فإن المحارف الموجودة بين القوسين تعني عدم وجود أحد

تلك المحارف في ذاك الموضع. لإزالة الغموض، جرّب المثال الآتي المُعدّل من المثال السابق:

```
[me@linuxbox ~]$ grep -h '[^bg]zip' dirlist*.txt
bunzip2
gunzip
funzip
gpg-zip
preunzip
prezip
prezip-bin
unzip
unzipsfx
unzipsfx
```

بعد تفعيل "الرفض"، ستظهر لنا قائمة تحتوي على كل الأسطر التي تحتوي على "zip" ويسبقها أي حرف ما عدا "b" و "g". لاحظ عدم ظهور zip في القائمة. المجموعة المُستخدمة في التعبير النظامي السابق ما تزال تتطلب وجود حرف في ذاك الموضع، لكنه (أي الحرف) ليس عضوًا في تلك المجموعة.

يعكس رمز "^" معنى المجموعة في حال كان أول محرف فيها؛ عدا ذلك فإنه يفقد معناه وُيعامَل على أنه حرف عادي.

مجالات الحروف التقليدية

إذا أردنا كتابة تعبير نظامي يُطابِق كل اسم في ملفاتنا السابقة يبدأ بحرف كبير، فإننا نستخدم الأمر الآتي:

```
[me@linuxbox ~]$ grep -h '^[ABCDEFGHIJKLMNOPQRSTUVWXZY]' dirlist*.txt
```

ليس من المنطقي كتابة 26 حرفًا في كل مرّة! هذه طريقة أخرى لذلك:

```
[me@linuxbox ~]$ grep -h '^[A-Z]' dirlist*.txt
MAKEDEV
ControlPanel
GET
HEAD
POST
X
X11
```

```
Xorg
MAKEFLOPPIES
NetworkManager
NetworkManagerDispatcher
```

باستخدام مجال حروف يتكون من ثلاثة محارف فقط، استطعنا تمثيل 26 حرفًا. أي مجال من المحارف (بما فيها الأرقام) يمكن استخدامه بهذه الطريقة. ويمكن أيضًا تحديد أكثر من مجال بين أقواس كالمثال الآتي الذي يُطابِق جميع أسماء الملفات التي تبدأ بأحرف أو أرقام:

```
[me@linuxbox ~]$ grep -h '^[A-Za-z0-9]' dirlist*.txt
```

نلاحظ أن الشرطة "-" تُعامَل معاملةً خاصةً، إذًا كيف نستطيع وضع الشرطة في تعبير الأقواس؟ يتم ذلك بوضعها كأول محرف في التعبير. جرّب المثالين الآتيين:

```
[me@linuxbox ~]$ grep -h '[A-Z]' dirlist*.txt
```

المثال السابق يُطابِق كل أسماء الملفات التي تحتوي على حرف كبير، بينما:

```
[me@linuxbox ~]$ grep -h '[-AZ]' dirlist*.txt
```

يُطابِق جميع أسماء الملفات التي تحتوي على الشرطة "-" أو حرف "A" أو "Z".

فئات حروف POSIX

مجالات المحارف التقليدية سهلة الفهم وطريقة فعالة لتحديد مجال من الحروف بسرعة. لكن لسوء الحظ، لا تعمل مجالات المحارف دائمًا عملًا صحيحًا. على الرغم من أننا لم نواجه مشاكل مع grep حتى الآن، لكن قد تحدث لنا مشاكل عند استخدام البرامج الأخرى.

بالعودة إلى الفصل الرابع، كنا قد ألقينا نظرةً على الحروف البديلة وكيفية استخدامها للقيام بتوسعة أسماء الملفات. في ذاك النقاش، قلنا أن مجالات الحروف تعمل بنفس الطريقة التي تعمل بها في التعابير النظامية، لكن هاك المشكلة:

```
[me@linuxbox ~]$ ls /usr/sbin/[ABCDEFGHIJKLMNOPQRSTUVWXYZ]*
/usr/sbin/MAKEFLOPPIES
/usr/sbin/NetworkManagerDispatcher
/usr/sbin/NetworkManager
```

(ربما تظهر لك قائمة مختلفة حسب توزيعتك وربما لا يظهر أي ناتج. جُرِّبَ هذا المثال على توزيعة أوبنتو).
يُظهِر الأمر السابق النتائج المتوقعة: قائمة بالملفات التي يبدأ اسمها بحرف كبير، لكن:

```
[me@linuxbox ~]$ ls /usr/sbin/[A-Z]*
/usr/sbin/biosdecode
/usr/sbin/chat
/usr/sbin/chgpasswd
/usr/sbin/chpasswd
/usr/sbin/chroot
/usr/sbin/cleanup-info
/usr/sbin/complain
/usr/sbin/console-kit-daemon
```

لقد ظهرت لنا نتيجة مختلفة تمامًا! (عُرِضَ جزء يسير من الناتج)، لماذا؟ إنها قصة طويلة، لكن هاك نسخةً مختصرةً منها:

بالعودة إلى الزمن الذي طوِّرَ يونكس فيه لأول مرّة، كان النظام يعرف فقط محارف ASCII، وكانت ميزاته تعكس هذه الحقيقة. أول 32 محرفًا (التي تقابل الأرقام من 0 إلى 31) هي أكواد التحكم (كمسافة الجدولة [tab] والفراغ الخلفي [backspace] ومحرف العودة إلى بداية السطر [carriage return]). تحتوي المحارف 32 التالية (32-63) على المحارف الطباعية، بما فيها أغلب علامات الترقيم والأرقام من 0 إلى 9. تحتوي المحارف 32 التي تليها (64-95) على الأحرف الكبيرة (uppercase) وبعض علامات الترقيم الأخرى. آخر 31 محرفًا (الأرقام من 96 إلى 127) تحتوي على الأحرف الصغيرة (lowercase) والمزيد من علامات الترقيم. لذا، فإن ASCII يرتب الأحرف بالشكل الآتي:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

والذي يختلف عن ترتيب المعاجم والقواميس، الذي يكون بالعادة كالآتي:

aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ

ظهرت الحاجة إلى دعم الأحرف غير الموجودة بعد أن انتشر يونكس خارج الولايات المتحدة الأميركية. وسِّعَ جدول ASCII لكي يستخدم ثماني بتات، وأضاف محارف للأرقام من 128-255، التي ضمت العديد من اللغات. لدعم هذه الإمكانية، قدّمت معايير POSIX مفهومًا جديدًا سُمِّي "المحليّة" (locale). التي يمكن أن تُعدَّل لتحديد مجموعة المحارف المستخدمة في مكان معين من العالم. بإمكاننا عرض قيمة متغير اللغة في نظامنا بالأمر الآتي:

```
[me@linuxbox ~]$ echo $LANG
```

```
en_US.UTF-8
```

ستستخدم البرمجيات التي تتبع معيار POSIX ترتيب أحرف يختلف عن ترتيب ASCII بالاعتماد على قيمة هذا المتغير. وهذا ما يفسر سلوك الأوامر التي جرّبناها في الأعلى. يُفسَّر مجال المحارف [A-Z] في ترتيب القاموس على أنه جميع الحروف الأبجدية ما عدا "a" بالحالة الصغيرة.

يمكن الالتفاف على هذه الإشكالية؛ حيث يحتوي معيار POSIX عددًا من فئات الحروف التي توفر مجالات مختلفة:

الجدول 19-2: فئات حروف POSIX

فئة الحروف	الشرح
[:alnum:]	جميع الحروف الأبجدية والأرقام. يمكن التعبير في ASCII عنها بالشكل: [A-Za-z0-9].
[:word:]	كما في الفئة [:alnum:] لكن مع إضافة الشرطة السفلية.
[:alpha:]	الأحرف الأبجدية. يمكن التعبير في ASCII عنها بالشكل: [A-Za-z].
[:blank:]	تتضمن الفراغ ومسافة الجدولة.
[:cntrl:]	أكواد التحكم في ASCII. تتضمن محارف ASCII ذات الأرقام من 0 إلى 31 و 127.
[:digit:]	الأرقام من 0 إلى 9.
[:graph:]	جميع المحارف المرئية. في ASCII هي المحارف ذات الأرقام من 33 إلى 126.
[:lower:]	جميع الأحرف الصغيرة.
[:punct:]	جميع علامات الترقيم. أي [-!"#$%&'()*+,./:;<=>?@[\\\]_`{|}~].
[:print:]	جميع المحارف الطباعية بما فيها تلك الموجود في الفئة [:graph:] بالإضافة إلى الفراغ.
[:space:]	المحارف الفاصلة بما فيها الفراغ ومسافة الجدولة والعودة إلى بداية السطر والسطر الجديد ومسافة الجدولة العمودية ومحرف form feed يُعبَّر عنهم في ASCII: [\t\r\n\v\f]

[:upper:]	الأحرف الكبيرة.
[:xdigit:]	المحارف المُستخدمة للتعبير عن الأرقام الست عشرية. تُمثَّل في ASCII بالشكل الآتي: [0-9A-Fa-f].

لا يمكن التعبير عن مجال جزئي ([A-M])، حتى باستخدام فئات الحروف.

سنكرر المثال السابق، لكن هذه المرّة مع استخدام فئات الحروف:

```
[me@linuxbox ~]$ ls /usr/sbin/[[:upper:]]*
/usr/sbin/NetworkManager
```

تذكر أن المثال السابق ليس مثالًا عن التعابير النظامية، بل مجرد توسعة أسماء الملفات التي تقوم بها الصدفة. أُستخدم هنا لأن بالإمكان استخدام فئات حروف POSIX لكلا الغرضَين.

العودة إلى الترتيب التقليدي

بإمكانك اختيار أن يَستخدِم نظامك ترتيب الحروف التقليدي (ASCII) بتعديل قيمة متغير البيئة LANG. كما شاهدنا في الفقرة السابق، يحتوي المتغير LANG على اسم اللغة ومجموعة المحارف المستخدمة. تُحدَّد القيمة الافتراضية لهذا المتغير أثناء اختيارك للغة أثناء تثبيت لينُكس.

لمشاهدة الضبط الخاص بـ"المحليّة"، استخدم الأمر locale:

```
[me@linuxbox ~]$ locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"

LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
```

أسند القيمة POSIX إلى المتغير LANG لتغيير المحليّة لاستخدام سلوك يونكس التقليدي:

```
[me@linuxbox ~]$ export LANG=POSIX
```

لاحظ أن هذا التغيير يجعل النظام يستخدم مجموعة المحارف الإنكليزية الخاصة بالولايات المتحدة الأميركية (محارف ASCII بالتحديد). لذا، تأكد إن كان ذلك ما تريد.

يمكنك جعل هذا التغيير ثابتًا بإضافة السطر الآتي إلى ملف `.bashrc` الخاص بك:

```
export LANG=POSIX
```

التعابير النظامية الأساسية في مواجهة التعابير النظامية الموسعة

عندما ظننا أن الأمر لن يكون مربكًا أكثر من هذا؛ فنفاجئ بأن معيار POSIX يقسّم التعابير النظامية إلى قسمين: التعابير النظامية الأساسية (basic regular expressions اختصارًا BRE)، والتعابير النظامية الموسعة (extended regular expressions اختصارًا ERE). الميزات التي ناقشناها حتى الآن هي مدعومة من أي تطبيق متوافق مع POSIX ويدعم التعابير النظامية الأساسية. صديقنا `grep` هو أحدهم.

ما هو الفرق ما بين التعابير النظامية الأساسية والموسعة؟ القضية هي قضية الحروف الخاصة. يمكن استخدام الأحرف الخاصة الآتية في التعابير النظامية الأساسية:

```
^ $ . [ ] *
```

أي محرف آخر يُعتبَر حرفًا عاديًا. لكن في التعابير النظامية الموسعة، أُضيفت الأحرف الآتية:

```
( ) { } ? + |
```

لكن (وها هو الأمر المسلي)، تُفسَّر الرموز "(" و ")" و "{" و "}" على أنها أحرف خاصة في التعابير النظامية الأساسية إذا تم تهريبهم بواسطة الشرطة المائلة الخلفية، لكن مع التعابير النظامية الموسعة، يُعتبَر أي حرف خاص حرفًا عاديًا عندما يُسبَق بالشرطة المائلة الخلفية. سيوضَّح أي التباس حدث في الفقرات الآتية.

ولأن المزايا التي سنناقشها في الفقرات اللاحقة هي جزء من التعابير النظامية الموسعة، فسنحتاج إلى استخدام نسخة أخرى من `grep`. تقليديًا، كان يتم ذلك باستخدام `egrep`؛ لكن نسخة غنو من برنامج `grep` تدعم التعابير النظامية الموسعة عند استخدام الخيار `-E`.

POSIX

خلال الثمانيات من القرن الماضي، أصبح يونكس من أشهر أنظمة التشغيل التجارية، لكن في عام 1988 حدثت فوضى في عالم يونكس. العديد من صانعي العتاد الذي حصلوا على رخصة الكود

المصدري لنظام يونكس من مالكيه، AT&T، زودوا عتادهم بنسخٍ مختلفة من النظام. لكن، وبسبب جهودهم في إحداث فروق للمنافسة، أضافت كل شركة مصنعة تغيرات مملوكة وإضافات خاصة بها إلى النظام. وهذا ما قلل من توافقية البرمجيات. وكما مع جميع الشركات التجارية، حاولت كل شركة ربح لعبة "كسب" الزبائن الدائمين.

في منتصف الثمانيات، بدأ IEEE (معهد مهندسي الكهرباء والإلكترون "Institute of Electrical and Electronics Engineers") تطوير معايير تُحدِّد كيف يجب على نظام يونكس (والأنظمة الشبيهة بِيونكس) أن يعمل. هذه المعايير، التي تُعرَف رسميًا بمعيار IEEE 1003، تُعرِّف واجهات برمجة التطبيقات (application programming interfaces) يُرمَز لها اختصارًا "API"، والصدفة والأدوات التي يجب أن تتوفر في النظام الشبيه بِيونكس القياسي. أُشتقت الكلمة "POSIX" من الجملة "Portable Operating System Interface" (أُضيف الحرف "X" لتسهيل اللفظ) وأُقترِحت من قِبل ريتشارد ستالمان (نعم، ريتشارد ستالمان ذاته) وأُستخدمت بعد ذلك من قِبل IEEE.

الاختيار

أول ميزة من ميزات التعابير النظامية الموسعة التي سنناقشها هي "الاختيار" (alternation)، وهي الآلية التي تسمح بمطابقة أحد الأنماط الموجودة في مجموعة من التعابير، بآلية تشبه عمل تعبيرات الأقواس التي تسمح لنا باختيار أحد المحارف الموجودة في المجموعة. الاختيار يسمح بأن تتم عملية المطابقة من مجموعة من السلاسل النصية أو التعابير النظامية الأخرى.

لإزالة ما سببته الفقرة السابقة من غموض؛ سنستخدم grep مع echo:

```
[me@linuxbox ~]$ echo "AAA" | grep AAA
AAA
[me@linuxbox ~]$ echo "BBB" | grep AAA
[me@linuxbox ~]$
```

مثال سهل جدًا ومباشر، حيث مررنا ناتج echo عبر الأنبوب إلى grep؛ وعند وجود مطابقة، فسوف نشاهد النتائج. إن لم يكن هناك مطابقة، فلن نشاهد أيّة نتائج!

الآن، حان وقت إضافة الاختيار، الذي يُحَدَّد باستخدام الخط الشاقولي "|":

```
[me@linuxbox ~]$ echo "AAA" | grep -E 'AAA|BBB'
AAA
```

```
[me@linuxbox ~]$ echo "BBB" | grep -E 'AAA|BBB'
BBB
[me@linuxbox ~]$ echo "CCC" | grep -E 'AAA|BBB'
[me@linuxbox ~]$
```

لاحظنا في المثال السابق وجود التعبير 'AAA|BBB' الذي يعني "إما طابق السلسلة النصية AAA أو السلسلة النصية BBB". ولأن هذه الميزة موجودة في التعابير النظامية الموسعة، فلاحظ استخدامنا للخيار E- في grep (على الرغم من استطاعتنا أن نشغل egrep بدلًا من ذلك)، ووضعنا التعبير النظامي بين علامتي اقتباس كي لا تُفسره الصدفة على أنه أنبوب. الاختيار غير محدود لخيارَين فقط:

```
[me@linuxbox ~]$ echo "AAA" | grep -E 'AAA|BBB|CCC'
AAA
```

نستخدم () لفصل ميزة الاختيار عن باقي عناصر التعابير النظامية:

```
[me@linuxbox ~]$ grep -Eh '^(bz|gz|zip)' dirlist*.txt
```

ستطابق هذه التوسعة أسماء الملفات في القوائم التي أنشأناها التي تبدأ بالكلمة "bz" أو "gz" أو "zip". إذا أزلنا القوسين، فسيتغير معنى التعبير النظامي:

```
[me@linuxbox ~]$ grep -Eh '^bz|gz|zip' dirlist*.txt
```

إلى مطابقة أي اسم ملف يبدأ بالكلمة "bz" أو يحتوي على "gz" أو "zip".

محددات التكرار

تدعم التعابير النظامية الموسعة عدّة طرق لتحديد عدد المرات التي سيُطابَق فيها عنصر ما.

الرمز "?": مطابقة العنصر "صفر" مرّة أو مرّة واحدة

هذا المحدد يعني "اجعل العنصر الذي يسبق هذا الرمز اختياريًا". لنفرض أنا نريد التحقق من صلاحية أرقام الهاتف. يحدد فيما إذا كان رقم الهاتف صحيحًا بمطابقته لأحد الشكلَين الآتيَين:

```
(nnn) nnn-nnnn
nnn nnn-nnnn
```

حيث "n" هو رقم من 0 إلى 9. يمكننا إنشاء التعبير الآتي:

```
^\(?[0-9][0-9][0-9]\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$
```

في التعبير السابق، أضفنا بعد الأقواس علامة الاستفهام "?" والتي تعني أن الأقواس يمكن أن لا تطابق أبدًا أو تطابق مرّة واحدة فقط. ولأن الأقواس تعتبر من الحروف الخاصة في ERE؛ فقد تم إسباقها بشرطة مائلة خلفية لكي تعامل معاملة الأحرف العادية.

لنجرب التعبير السابق:

```
[me@linuxbox ~]$ echo "(555) 123-4567" | grep -E '^\(?[0-9][0-9][0-9]\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'
(555) 123-4567
[me@linuxbox ~]$ echo "555 123-4567" | grep -E '^\(?[0-9][0-9][0-9]\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'
555 123-4567
[me@linuxbox ~]$ echo "AAA 123-4567" | grep -E '^\(?[0-9][0-9][0-9]\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'
[me@linuxbox ~]$
```

لاحظ كيف طابق النمط شكلَيّ أرقام الهاتف الصحيحَين لكنه لم يطابق الشكل الذي يحتوي على قيم غير عددية.

الرمز "*": مطابقة العنصر "صفر" مرّة أو أكثر

وكما في الرمز "?"؛ يستخدم الرمز "*" لجعل العنصر الذي يسبقه اختياريًا، لكن يمكن لذاك العنصر أن يُطابَق لأي عدد من المرات وليس لمرّة واحدة فقط كما في الرمز "?". لنفترض أننا نريد معرفة فيما إذا كانت سلسلةٌ نصيةٌ ما هي جملة؛ أي أنها تبدأ بحرف كبير وتحتوي على أي عدد من الحروف الكبيرة والصغيرة والفراغات، ثم تنتهي بنقطة. لمطابقة هذا الوصف (البسيط) للجملة، فإننا نستخدم التعبير النظامي الآتي:

```
[[:upper:]][[:upper:][:lower:] ]*\.
```

يتكون التعبير السابق من ثلاثة عناصر: تعبير أقواس يحتوي على نمط الحروف [:upper:]، وتعبير أقواس آخر يحتوي على فئات الحروف [:upper:] و [:lower:] بالإضافة إلى الفراغ، ورمز النقطة مسبوقًا بشرطة مائلة خلفية لكي تُعامَل النقطة كالأحرف العادية. لاحظ أن العنصر الثاني يأتي بعده الرمز "*". إذًا، بعد وجود الحرف الكبير في أول الجملة يطابق أي عدد من الحروف الكبيرة والصغيرة والفراغات، ومن ثم تكون النقطة في آخر الجملة:

```
[me@linuxbox ~]$ echo "This works." | grep -E '[[:upper:]][[:upper:]
```

```
[:lower:] ]*\.'
This works.
[me@linuxbox ~]$ echo "This Works." | grep -E '[[:upper:]][[:upper:]
[:lower:] ]*\.'
This Works.
[me@linuxbox ~]$ echo "this does not" | grep -E '[[:upper:]][[:upper:]
[:lower:] ]*\.'
[me@linuxbox ~]$
```

طابق التعبير أول اختبارين، لكنه لم يطابق الثالث، بسبب نقصان النقطة في آخر الجملة والحرف الكبيرة في أولها.

الرمز "+": مطابقة العنصر مرّة واحدة أو أكثر

يعمل الرمز "+" بشكل مشابه للرمز "*"، إلا أنه يتطلب مطابقة واحدة على الأقل للعنصر الذي يسبق هذا الرمز. يطابق التعبير الآتي الأسطر التي تتألف من مجموعات تتكون من حرف واحد أو أكثر يفصل بين تلك المجموعات فراغ واحد فقط:

```
^([[:alpha:]]+ ?)+$
```

مثال:

```
[me@linuxbox ~]$ echo "This that" | grep -E '^([[:alpha:]]+ ?)+$'
This that
[me@linuxbox ~]$ echo "a b c" | grep -E '^([[:alpha:]]+ ?)+$'
a b c
[me@linuxbox ~]$ echo "a b 9" | grep -E '^([[:alpha:]]+ ?)+$'
[me@linuxbox ~]$ echo "abc  d" | grep -E '^([[:alpha:]]+ ?)+$'
[me@linuxbox ~]$
```

لاحظنا أن التعبير السابق لم يُطابق السطر "a b 9" لأنه يحتوي على حرف غير أبجدي (الرقم 9). ولم يطابق السطر "abc d" أيضًا بسبب فصل أكثر من فراغ بين الحرفين "c" و "d".

الرمز "{}": مطابقة العنصر لعدد محدد من المرات

يُستخدم الحرفَين الخاصَين "{" و "}" للتعبير عن العدد الأدنى والعدد الأعلى من المطابقات المطلوبة للعنصر الذي يسبقهما. نستطيع التعبير عن ذلك بأربع طرق:

الجدول 19-3: تحديد عدد المطابقات

المحدد	مطابقة العنصر السابق إذا تكرر
`{n}`	n مرة فقط.
`{n, m}`	n مرة على الأقل ولكن ليس أكثر من m مرة.
`{n,}`	n مرة على الأقل.
`{,m}`	m مرة على الأكثر.

بالعودة إلى مثال أرقام الهواتف السابق، يمكننا استخدام هذه الطريقة لتحديد التكرارات وتبسيط شكل التعبير النظامي:

```
^\(?[0-9][0-9][0-9]\)?  [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$
```

إلى:

```
^\(?[0-9]{3}\)?  [0-9]{3}-[0-9]{4}$
```

لنجرب ذلك:

```
[me@linuxbox ~]$ echo "(555) 123-4567" | grep -E '^\(?[0-9]{3}\)? [0-9]
{3}-[0-9]{4}$'
(555) 123-4567
[me@linuxbox ~]$ echo "555 123-4567" | grep -E '^\(?[0-9]{3}\)? [0-9]
{3}-[0-9]{4}$'
555 123-4567
[me@linuxbox ~]$ echo "5555 123-4567" | grep -E '^\(?[0-9]{3}\)? [0-9]
{3}-[0-9]{4}$'
[me@linuxbox ~]$
```

كما لاحظتَ، يُطابِق التعبير النظامي السابق أرقام الهواتف الصحيحة بنجاح (التي تحتوي على أقواس، والتي لا تحتويها)، ويرفض أرقام الهواتف ذات الشكل الخاطئ.

استخدامات عملية للتعابير النظامية

لنلقِ نظرة على بعض الأوامر التي نعرفها مسبقًا ولنتعلم كيف يمكن استخدامها مع التعابير النظامية.

التحقق من صحة قائمة أرقام هواتف باستخدام grep

في مثالنا السابق، جرّبنا التحقق من صحة رقم هاتف واحد فقط. التحقق من قائمة أرقام وليس رقم واحد فقط هو مثال أكثر واقعيةً. لذا، لننشئ قائمتنا. سنُنفّذ أمرًا "سحريًا" (هو كذلك لأنك لم تتعلم بعد الأوامر المُستخدمة فيه. لكن لا تقلق، سنشرح آلية عمله بالتفصيل في الفصول القادمة):

```
[me@linuxbox ~]$ for i in {1..10}; do echo "(${RANDOM:0:3})
${RANDOM:0:3}-${RANDOM:0:4}" >> phonelist.txt; done
```

سيولد الأمر السابق قائمةً باسم phonelist.txt تحتوي على عشرة أرقام هواتف. ستُضاف عشرة أرقام أخرى إلى الملف في كل مرّة يُنفَّذ الأمر السابق فيها. يمكننا أيضًا تغيير القيمة 10 القريبة من بداية الأمر لتوليد عدد أقل أو أكثر من أرقام الهواتف. إذا تفحصنا محتوى ملف phonelist.txt، فسنجد مشكلةً:

```
[me@linuxbox ~]$ cat phonelist.txt
(232) 298-2265
(624) 381-1078
(540) 126-1980
(874) 163-2885
(286) 254-2860
(292) 108-518
(129) 44-1379
(458) 273-1642
(686) 299-8268
(198) 307-2440
```

بعض الأرقام غير صالحة، ولكن هذا ما يجعل القائمة السابقة ممتازة للتجربة عليها، لأننا سنستخدم grep للتحقق من صِحتهم.

طريقة مفيدة للتحقق من الأرقام هي البحث في الملف عن جميع الأرقام غير الصحيحة وإظهارهم على الشاشة:

```
[me@linuxbox ~]$ grep -Ev '^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$'
phonelist.txt
(292) 108-518
(129) 44-1379
[me@linuxbox ~]$
```

استخدمنا الخيار "v-" للحصول على السطور التي لا تحتوي على مطابقات. يحتوي التعبير على محرفَيّ بداية السطر "^" ونهاية السطر "$" للتأكد من أن السطر لا يحتوي إلا على رقم الهاتف دون أيّة بيانات أُخرى (فراغات في بداية أو نهاية السطر). ولاحظ أيضًا أن التعبير يتطلب وجود الأقواس، على عكس المثال الأسبق الذي كانت فيه الأقواس اختيارية.

العثور على أسماء الملفات غير المحبذة باستخدام find

يدعم البرنامج find اختبارًا يستخدم التعابير النظامية. هنالك شيء مهم يجب أخذه بعين الاعتبار هو أن طريقة استخدام التعابير النظامية في find تختلف عن grep. يطبع grep السطر عندما يحتوي على النمط، أما find، فيتطلب أن يكون اسم الملف مُطابِقًا تمامًا لنمط التعابير النظامية. في المثال الآتي، سنستخدم الأمر find مع التعابير النظامية للبحث عن كل ملف لا يحتوي اسمه على مجموعة المحارف الآتية:

```
[-_./0-9a-zA-Z]
```

سيُظهِر مثل هذا البحث أسماء الملفات التي تحتوي على فراغات وغيرها من المحارف غير المُحبذ استخدامها في أسماء الملفات:

```
[me@linuxbox ~]$ find . -regex '.*[^-_./0-9a-zA-Z].*'
```

ولأن find يتطلب أن يُطابِق التعبير النظامي اسم الملف بالكامل، فقد وضعنا "*." في بداية ونهاية التعبير للمطابقة صفر مرّة أو أكثر لأي محرف. استخدمنا في وسط التعبير مجموعةً مرفوضةً من جميع المحارف التي ليست "غير محبذة" الاستخدام في أسماء الملفات.

البحث عن الملفات باستخدام locate

يدعم برنامج locate التعابير النظامية العادية (باستخدام الخيار regexp--) والموسعة (بالخيار regex--). بإمكاننا إجراء نفس العمليات التي نَفّذناها على ملف dirlist السابق:

```
[me@linuxbox ~]$ locate --regex 'bin/(bz|gz|zip)'
/bin/bzcat
/bin/bzcmp
/bin/bzdiff
/bin/bzegrep
/bin/bzexe
/bin/bzfgrep
/bin/bzgrep
```

```
/bin/bzip2
/bin/bzip2recover
/bin/bzless
/bin/bzmore
/bin/gzexe
/bin/gzip
/usr/bin/zip
/usr/bin/zipcloak
/usr/bin/zipgrep
/usr/bin/zipinfo
/usr/bin/zipnote
/usr/bin/zipsplit
```

باستخدام خاصية "الاختيار"، بحثنا عن الملفات التي تحتوي على "bin/bz" أو "bin/gz" أو "bin/zip".

البحث عن النصوص في less و vim

يتشارك برنامجَيّ less و vim في طريقة البحث عن النصوص. سيؤدي الضغط على / وإدخال نمط التعابير النظامية إلى بدء البحث. إذا استخدمنا less لعرض ملف phonelist.txt السابق:

```
[me@linuxbox ~]$ less phonelist.txt
```

ومن ثم أدخلنا نمط التعابير النظامية:

```
(232) 298-2265
(624) 381-1078
(540) 126-1980
(874) 163-2885
(286) 254-2860
(292) 108-518
(129) 44-1379
(458) 273-1642
(686) 299-8268
(198) 307-2440
~
~
~
```

```
/^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$
```

فسيُعَلِّم less السلاسل النصية التي تطابق النمط:

```
(232)   298-2265
(624)   381-1078
(540)   126-1980
(874)   163-2885
(286)   254-2860
(292)   108-518
(129)   44-1379
(458)   273-1642
(686)   299-8268
(198)   307-2440
~
~
~
(END)
```

من ناحية أُخرى، يدعم vim التعابير النظامية العادية، لذا، سيكون نمط البحث بالشكل الآتي:

```
/([0-9]\{3\}) [0-9]\{3\}-[0-9]\{4\}
```

يمكنك ملاحظة أن النمط مشابه للنمط السابق إلا أن بعض المحارف التي تعتبر حروفًا خاصة في التعابير النظامية الموسعة تعامل كحروف عادية في التعابير النظامية العادية. لكنها ستُعامل كأحرف خاصة عند إسباقها بشرطة مائلة خلفية. سيُعَلِّم النص المُطابق للبحث وذلك بالاعتماد على نسخة vim المُثبّتة على نظامك. إذا لم يتم ذلك، جرّب هذا الأمر في وضع الأوامر:

```
:hlsearch
```

لتفعيل تعليم مطابقات البحث.

ملاحظة: ربما يدعم vim البحث عن النصوص وربما لا يدعمها، وذلك بالاعتماد على التوزيعة التي تستخدمها. توفر أوبنتو على سبيل المثال، نسخةً مُصغّرةً من vim افتراضيًا. في مثل هذه التوزيعات، يمكنك استخدام مدير الحزم لتثبيت الحزمة الكاملة من vim.

الخلاصة

تعلمنا في هذا الفصل العديد من استخدامات التعابير النظامية. يمكننا الحصول على المزيد من الاستخدامات بالبحث عن البرمجيات الأخرى التي تدعمهم. يمكننا معرفة تلك البرمجيات بالبحث في صفحات الدليل man:

```
[me@linuxbox ~]$ cd /usr/share/man/man1
[me@linuxbox man1]$ zgrep -El 'regex|regular expression' *.gz
```

يمكن استخدام برنامج zgrep بنفس آلية استخدام برنامج grep لكن لقراءة الملفات النصية المضغوطة. بحثنا، في المثال السابق، في ملفات القسم الأول المضغوطة من الدليل man الموجودة في موضعهم الاعتيادي. حيث ستظهر قائمة بالملفات التي تحتوي إحدى العبارتين: "regex" أو "regular expression" كنتيجة للأمر السابق. كما ستلاحظ، تُستخدم التعابير النظامية في العديد من البرامج.

هنالك ميزة في التعابير النظامية العادية لم نشرحها بعد. تسمى "الأنماط الفرعية". سنشرح هذه الميزة في الفصل القادم.

الفصل العشرون:

معالجة النصوص

تعتمد جميع الأنظمة الشبيهة بِيونكس على النصوص اعتمادًا كبيرًا لتخزين مختلف أنواع البيانات. وهذا ما يفسر امتلاك تلك الأنظمة العديد من أدوات تعديل النصوص. سنلقي في هذا الفصل نظرةً على الأدوات التي تستخدم "لتفريق" و"تجميع" النصوص. وسنلقي نظرة في الفصل القادم على المزيد من أدوات معالجة النصوص، مركزين على البرامج التي تستخدم لتنسيق النص لغرض طباعته أو لغيره من الأغراض.

سنزور في هذا الفصل بعض الأصدقاء القدامى وسنتعرف على آخرين جُدد:

- `cat` – دمج أو لمّ الملفات وطباعتها إلى مجرى الخرج القياسي.
- `sort` – ترتيب أسطر ملفٍ نصي.
- `uniq` – التبليغ عن أو حذف الأسطر المكررة.
- `cut` – إزالة أقسام من أسطر ملفٍ ما.
- `paste` – دمج أسطر عدّة ملفات.
- `join` – دمج أسطر ملفين.
- `comm` – المقارنة بين ملفَين مُرتبَين سطرًا بسطر.
- `diff` – المقارنة بين ملفَين سطرًا بسطر.
- `patch` –تحويل ملف ذي نسخة قديمة إلى نسخة أجدد عن طريق ملف الاختلافات `diff`.
- `tr` – تحويل أو حذف المحارف.
- `sed` – محرر تدفقي لترشيح (فلترة) أو تحويل النصوص.
- `aspell` – مدقق إملائي تفاعلي.

مجالات استخدام النصوص

تعلمنا، حتى الآن، استخدام محررَين نصيَين (nano و `vim`)، وألقينا نظرةً على العديد من ملفات الإعدادات، وكنا أيضًا شاهدين على مخرجات عشرات الأوامر، وكل ذلك كان عبارة عن "نص". لكن هل هنالك استخدامات أُخرى للنصوص؟ نعم، يوجد الكثير!

المستندات

يكتب العديد من الأشخاص مستنداتهم كنصوص عادية. بينما من السهل تخيل استخدام ملف نصي بسيط وصغير لتخزين الملاحظات أو المذكرات، إلا أنه بالإمكان أيضًا كتابة مستندات كبيرة في الصيغة النصية. إحدى الطرق المشهورة هي كتابة المستندات الضخمة في صيغة نصية بسيطة ومن ثم استخدام لغة وصفية (markup language) لشرح هيئة وتنسيق المستند النهائي. كُتِبَت العديد من الأبحاث العلمية بهذه الطريقة، لأن نظم معالجة النصوص المستخدمة في يونكس هي من أولى النظم التي تدعم التخطيطات الطباعية المعقدة التي كان يحتاجها الكُتّاب.

صفحات الوِب

ربما أشهر أنواع المستندات الرقمية الموجودة في العالم هي صفحة الوِب. صفحات الوِب هي مستندات نصية تكون إما بصيغة HTML (Hypertext Markup Language) أو XML (Extensible Markup Language) التي تستخدم لوصف هيئة المستند المرئية.

البريد الإلكتروني

يعتمد البريد الإلكتروني في جوهره على النصوص. حتى الملفات المرفقة غير النصية تُحوَّل إلى صيغة نصية لكي تُنقل عبر الشبكة. يمكننا التأكد من ذلك بأنفسنا بتنزيل رسالة إلكترونية ومشاهدتها في `less`. سنلاحظ أن الرسالة تبدأ بالترويسة (Header) التي تحتوي على معلومات حول مصدر الرسالة والمعالجة التي أُجريت عليها حتى وصلت إلينا، ومن ثم يتبعها الجسم (body) الذي يشكل محتوى الرسالة.

الطباعة

يُرسَل الخرج الموجه للطابعة في الأنظمة الشبيهة بِيونكس على شكل نص عادي، وإذا حوى المستند على رسومات، فستحوَّل الرسومات إلى صيغة نصية هي لغة وصف الصفحات (page description language) المعروفة بالمصطلح PostSctipt، ومن ثم تُرسَل الصيغة النصية إلى برنامج يولّد النقط التي يجب على الطابعة طباعتها.

الكود المصدري للبرامج

أُنشئت العديد من أدوات سطر الأوامر الموجودة في الأنظمة الشبيهة بِيونكس بغرض مساعدة مدراء الأنظمة ومطوري البرمجيات؛ وأدوات معالجة النصوص ليست استثناءً لهذه القاعدة. العديد من تلك الأدوات موجه لحل مشاكل تطوير البرمجيات. السبب في أن معالجة النصوص هي مهمة لدرجة كبيرة لمطوري البرمجيات هو أن أصل جميع البرامج عبارة عن نصوص. القسم الذي يكتبه المبرمج (يُسمى "الكود المصدري") هو دائمًا

في صيغة نصية بسيطة.

زيارة أصدقائنا القدامى

بالعودة إلى الفصل السادس (إعادة التوجيه)، تعلمنا بعض الأوامر التي تقبل المدخلات من مجرى الدخل القياسي بالإضافة إلى الملفات الممررة كوسائط. شرحنا في ذاك النقاش عملهم شرحًا سطحيًّا، لكن حان الوقت الآن لإلقاء نظرة معمقة على طريقة عملهم وكيفية استخدامهم لمعالجة النصوص.

cat

لدى البرنامج cat عدد من الخيارات المثيرة للاهتمام. تساعد العديد من تلك الخيارات على "تخيل" محتوى النص بشكل أفضل. أحد الأمثلة على ذلك هو الخيار A-، الذي يُستخدم لعرض المحارف غير الطباعية في النص. هنالك حالات قد نحتاج فيها إلى عرض أكود التحكم الموجودة في النص. أحد أشهر تلك الأكواد هو "مفتاح الجدولة" (tab)، ومحرف العودة إلى بداية السطر (سنتعرف على سبب تسميته بهذا الاسم في فصلٍ لاحق)، الذي يمثل محرف نهاية السطر في الملفات النصية التي تستخدم نمط MS-DOS. مثال آخر هو الملفات التي تنتهي أسطرها بعدّة فراغات.

لنُنشئ ملفًا لكي نجرب عليه مستخدمين cat كمحرر نصي بسيط. وذلك بطباعة اسم الأمر فقط "cat" (طبعًا مع اسم الملف الذي ستوجه إليه المخرجات) ونطبع بعدها النص الذي نريد ثم نضغط على Enter ونطبع محرف نهاية الملف (EOF) بالضغط على Ctrl-d. أدخلنا في المثال الآتي مسافة جدولة عند أول السطر وأربعة فراغات عند نهايته:

```
[me@linuxbox ~]$ cat > foo.txt
	The quick brown fox jumped over the lazy dog.    
[me@linuxbox ~]$
```

سنستخدم الآن الأمر cat مع الخيار A- لعرض الملف الناتج:

```
[me@linuxbox ~]$ cat -A foo.txt
^IThe quick brown fox jumped over the lazy dog.    $
[me@linuxbox ~]$
```

كما لاحظنا من الناتج، قد تم تمثيل مفتاح الجدولة بالرمز "I^"، هذا النوع من الرموز شائع ويشير إلى "Control-I" (الذي هو نفسه زر tab). وشاهدنا أيضًا الرمز "$" في نهاية السطر للإشارة إلى أن السطر يحتوي في نهايته على أربعة فراغات.

نصوص MS-DOS في مواجهة نصوص يونكس

أحد الأسباب التي تجعلنا نستخدم cat للعثور على المحارف غير الطباعية في النص هو محاولة إيجاد محارف العودة إلى بداية السطر "المختبئة". من أي تأتي محارف العودة إلى بداية السطر؟ من DOS وويندوز! لا يُعرِّف يونكس و DOS نهاية السطر بنفس الطريقة. يُنهي يونكس السطر باستخدام محرف نهاية السطر (ASCII 10). أما MS-DOS، فإنه يُنهي السطر باستخدام محرف العودة إلى بداية السطر (ASCII 13) بالإضافة إلى محرف نهاية السطر.

هنالك عدّة طرق للتحويل من صيغة ملفات DOS إلى صيغة ملفات يونكس. يوجد في أغلب توزيعات لينُكس برنامجان باسم dos2unix و unix2dos، يسمحان بالتحويل من وإلى صيغة ملفات DOS. لكن لا تقلق إن لم يكن برنامج dos2unix مثبتًا على جهازك؛ فعملية تحويل صيغة الملفات النصية من DOS إلى يونكس سهلة جدًا، فقط احذف جميع محارف العودة إلى بداية السطر الموجودة في الملف. الأمر الذي يمكن القيام به بواسطة أداتين من الأدوات التي سنناقشها لاحقًا في هذا الفصل.

يملك cat عدّة خيارات لتعديل النصوص، أشهرها هو الخيار ‎-n‎ الذي يطبع أرقام الأسطر، و ‎-s‎ الذي يجعل cat لا يعرض الأسطر الفارغة:

```
[me@linuxbox ~]$ cat > foo.txt
The quick brown fox

jumped over the lazy dog.
[me@linuxbox ~]$ cat -ns foo.txt
       1        The quick brown fox
       2
       3        jumped over the lazy dog.
[me@linuxbox ~]$
```

أنشأنا في المثال السابق نسخةً جديدةً من ملف foo.txt الذي يحتوي على سطرين من النص يُفصَلُ بينهما بسطرين فارغين. لاحظ أن السطر الفارغ حُذف ورُقِّمت باقي الأسطر بعد معالجة الملف باستخدام الخيارَين ‎-ns‎.

sort

يرتب البرنامج sort محتويات مجرى الدخل القياسي أو ملف واحد أو أكثر، ويُرسِل المخرجات إلى مجرى الخرج القياسي. سنجرب معالجة مجرى الدخل القياسي من لوحة المفاتيح مباشرةً باستخدام نفس الطريقة

التي استخدمناها سابقًا مع cat:

```
[me@linuxbox ~]$ sort > foo.txt
c
b
a
[me@linuxbox ~]$ cat foo.txt
a
b
c
```

طبعنا الأحرف "c" و "b" و "a" (بعد إدخال الأمر السابق) ومن ثم Ctrl-d لإرسال محرف نهاية الملف. بعدها شاهدنا محتوى الملف الناتج ولاحظنا أن الأسطر قد رُتِّبَت.

ولأن sort يقبل أكثر من ملف كوسيط، فمن الممكن استخدامه لدمج الملفات في ملفٍ واحدٍ مرتب. على سبيل المثال، إذا كان لدينا ثلاثة ملفات وأردنا دمجها في ملفٍ واحد مرتب، فإننا نُطبق أمرًا شبيهًا بالأمر الآتي:

```
sort file1.txt file2.txt file3.txt > final_sorted_list.txt
```

يوجد العديد من الخيارات المثيرة للاهتمام للأمر sort. هذه قائمة جزئية منها:

الجدول 20-1: خيارات sort الشائعة

الخيار	الخيار الطويل	الشرح
-b	--ignore-leading-blanks	يُرتّب sort، افتراضيًا، بالاعتماد على السطر بأكمله، ويتم البدء من الحرف الأول من السطر. يؤدي هذا الخيار إلى جعل sort يتجاهل الفراغات الموجودة في أول السطر ويعتمد على أول محرف في السطر لا يكون فراغًا أو مسافة جدولة.
-f	--ignore-case	جعل عملية الترتيب غير حساسة لحالة الأحرف.
-n	--numeric-sort	الترتيب بالاعتماد على القيمة العددية. يؤدي استخدام هذا الخيار إلى جعل عملية الترتيب مُعتمدة على القيمة العددية بدلًا من القيمة الأبجدية (الترتيب الأبجدي للحروف).
-r	--reverse	عكس الترتيب. أي أن الناتج سيُرتَّب تنازليًا بدل ترتيبه تصاعديًا.

-k	--key=field1[,field2]	الـترتيب بالاعتمـاد علـى حقـل موجـود بيـن field1 و field2 بـدلًا مـن كـل السـطر. راجـع النقـاش فـي الأسـفل لمزيدٍ من المعلومات.
-m	--merge	معاملة كل وسيط كمسار ملف مرتب. ودمج الملفات المرتبة فـي ملـف واحـد مرتـب أيضًـا دون القيـام بعمليـة ترتيـب إضافية.
-o	--output=file	إرسال الناتج المرتب إلى الملف file بدلًا من مجرى الخرج القياسي.
-f	--field-separator=char	تحديد المحرف الفاصل ما بين الحقول. تفصل الفراغات أو مسافات الجدولة ما بين الحقول افتراضيًّا.

على الرغم من أن معظم الخيارات السابقة تشرح نفسها بنفسها؛ إلا أن بعضها الآخر ليس كذلك. لنلقِ أولًا نظرة على الخيـار n- الذي يجعل sort يرتـب بالاعتماد على القيـم العددية. يمكننا إزالة الغموض عن هذا الخيـار بتجربته مع ناتج الأمر du الذي يُستعمل لتحديـد أكبر المستخدمين للحجم التخزيني للقرص. يرتـب الأمر du الناتج حسب المسار افتراضيًّا:

```
[me@linuxbox ~]$ du -s /usr/share/* | head
252             /usr/share/aclocal
96              /usr/share/acpi-support
8               /usr/share/adduser
196             /usr/share/alacarte
344             /usr/share/alsa
8               /usr/share/alsa-base
12488           /usr/share/anthy
8               /usr/share/apmd
21440           /usr/share/app-install
48              /usr/share/application-registry
```

مررنا الناتج عبر الأنبوب إلى الأمر head لكي نحصل على عشرة أسطر فقط. نستطيع الترتيب حسب القيم العددية وإظهار أكثر عشرة عناصر مستهلكة للمساحة باستخدام الأمر الآتي:

```
[me@linuxbox ~]$ du -s /usr/share/* | sort -nr | head
```

```
509940          /usr/share/locale-langpack
242660          /usr/share/doc
197560          /usr/share/fonts
179144          /usr/share/gnome
146764          /usr/share/myspell
144304          /usr/share/gimp
135880          /usr/share/dict
76508           /usr/share/icons
68072           /usr/share/apps
62844           /usr/share/foomatic
```

استطعنا إنتاج قائمة معكوسة مرتبة عدديًا باستخدام الخيارَين nr-؛ حيث يظهر فيها أكبر القيم في أول الناتج. نجحت عملية الترتيب في المثال السابق لأن القيم العددية موجودة في أول كل سطر؛ لكن ماذا لو أردنا ترتيب القائمة بالاعتماد على قيمة ما موجودة في وسط السطر؟ على سبيل المثال ناتج الأمر ls -l:

```
[me@linuxbox ~]$ ls -l /usr/bin | head
total 152948
-rwxr-xr-x 1   root  root      34824  2008-04-04 02:42  [
-rwxr-xr-x 1   root  root     101556  2007-11-27 06:08  a2p
-rwxr-xr-x 1   root  root      13036  2008-02-27 08:22  aconnect
-rwxr-xr-x 1   root  root      10552  2007-08-15 10:34  acpi
-rwxr-xr-x 1   root  root       3800  2008-04-14 03:51  acpi_fakekey
-rwxr-xr-x 1   root  root       7536  2008-04-19 00:19  acpi_listen
-rwxr-xr-x 1   root  root       3576  2008-04-29 07:57  addpart
-rwxr-xr-x 1   root  root      20808  2008-01-03 18:02  addr2line
-rwxr-xr-x 1   root  root     489704  2008-10-09 17:02  adept_batch
```

تجاهل أننا نستطيع جعل الأمر ls يُرتِّب نتائجه حسب الحجم التخزيني، نستطيع استخدام sort لترتيب القائمة حسب حجم الملف:

```
[me@linuxbox ~]$ ls   -l /usr/bin | sort -nr -k 5 | head
-rwxr-xr-x 1   root  root    8234216 2008-04-07 17:42 inkscape
-rwxr-xr-x 1   root  root    8222692 2008-04-07 17:42 inkview
-rwxr-xr-x 1   root  root    3746508 2008-03-07 23:45 gimp-2.4
-rwxr-xr-x 1   root  root    3654020 2008-08-26 16:16 quanta
-rwxr-xr-x 1   root  root    2928760 2008-09-10 14:31 gdbtui
```

```
-rwxr-xr-x 1   root  root    2928756 2008-09-10 14:31 gdb
-rwxr-xr-x 1   root  root    2602236 2008-10-10 12:56 net
-rwxr-xr-x 1   root  root    2304684 2008-10-10 12:56 rpcclient
-rwxr-xr-x 1   root  root    2241832 2008-04-04 05:56 aptitude
-rwxr-xr-x 1   root  root    2202476 2008-10-10 12:56 smbcacls
```

العديد من استخدامات sort تكون لمعالجة البيانات المجدولة، كناتج الأمر ls في الأعلى. إذا طبّقنا مصطلحات قواعد البيانات على الجدول أعلاه، فإننا سنسمي كل سطر بالسجل (record). وكل سجل يحتوي على عدّة حقول (fields)، كخصائص الملف، وعدد الوصلات، واسم الملف، وحجمه التخزيني... يستطيع الأمر sort أن يعالج الحقول المختلفة. في مصطلحات قواعد البيانات، نستطيع تحديد حقل مفتاحي أو أكثر (key field) كمفاتيح الترتيب. استخدمنا، في المثال السابق، الخيارَين n و r للقيام ببحث عكسي مرتب عدديًا، والخيار k 5- لجعل sort يعتمد على الحقل الخامس (حجم الملف) في الترتيب.

الخيار k مثير للاهتمام ولديه عدّة ميزات، لكن يجب علينا أولًا شرح كيف يُعرِّف sort الحقول. سنفترض أنه لدينا الملف النصي البسيط الآتي:

```
William    Shotts
```

افتراضيًا، يَعتبر sort السطر السابق مكونًا من حقلين، أول حقل يحتوي على المحارف:

```
"William"
```

والحقل الثاني يحتوي على المحارف:

```
"    Shotts"
```

هذا يعني أن الفراغات ومسافات الجدولة تُستخدم للفصل ما بين الحقول، وإن هذه الفواصل ستُعتبر جزءًا من الحقل عند الترتيب.

لنلقِ نظرة أخرى على سطر ما من ناتج الأمر ls السابق؛ يمكننا أن نلاحظ أن السطر يحتوي على ثمانية حقول، الخامس منهم هو حجم الملف:

```
-rwxr-xr-x 1   root  root    8234216 2008-04-07 17:42 inkscape
```

في سلسلتنا التالية من التمارين، سنعتمد على الملف الآتي الذي يحتوي على تاريخ إصدارات ثلاث توزيعات لينُكس شهيرة بين عامَيّ 2006 إلى 2008. يحتوي كل سطر في الملف على ثلاثة حقول: اسم التوزيعة، ورقم الإصدار، وتاريخ الإصدار بالشكل MM/DD/YYYY:

```
SUSE          10.2    12/07/2006
Fedora        10      11/25/2008
SUSE          11.0    06/19/2008
Ubuntu        8.04    04/24/2008
Fedora        8       11/08/2007
SUSE          10.3    10/04/2007
Ubuntu        6.10    10/26/2006
Fedora        7       05/31/2007
Ubuntu        7.10    10/18/2007
Ubuntu        7.04    04/19/2007
SUSE          10.1    05/11/2006
Fedora        6       10/24/2006
Fedora        9       05/13/2008
Ubuntu        6.06    06/01/2006
Ubuntu        8.10    10/30/2008
Fedora        5       03/20/2006
```

سنحفظ هذه البيانات باستخدام محرر نصي (ربما vim) في ملف `distros.txt`. سنرتّب الآن الملف وسنشاهد الناتج:

```
[me@linuxbox ~]$ sort distros.txt
Fedora        10      11/25/2008
Fedora        5       03/20/2006
Fedora        6       10/24/2006
Fedora        7       05/31/2007
Fedora        8       11/08/2007
Fedora        9       05/13/2008
SUSE          10.1    05/11/2006
SUSE          10.2    12/07/2006
SUSE          10.3    10/04/2007
SUSE          11.0    06/19/2008
Ubuntu        6.06    06/01/2006
Ubuntu        6.10    10/26/2006
Ubuntu        7.04    04/19/2007
Ubuntu        7.10    10/18/2007
Ubuntu        8.04    04/24/2008
```

```
Ubuntu          8.10    10/30/2008
```

حسنًا، لقد أوشك على العمل عملًا صحيحًا؛ المشكلة حدثت في ترتيب إصدارات توزيعة فيدورا. لما كان الرقم "1" يأتي قبل "5" في الترتيب الأبجدي (وليس الترتيب العددي)، فإن الإصدار "10" سيكون في أعلى القائمة والإصدار "9" في آخرها.

لحل هذه المشكلة، سنحتاج إلى الترتيب بالاعتماد على أكثر من "مفتاح". نريد أن نجري ترتيبًا أبجديًا في أول حقل وترتيبًا عدديًا في الحقل الثاني. يسمح `sort` باستخدام الخيار k- أكثر من مرّة لكي يتم الترتيب بالاعتماد على أكثر من حقل مفتاحي. في الواقع، يمكن أن يكون المفتاح مجالًا من الحقول. إذا لم يحدد مجال (كما هو الحال في مثالنا السابق)، فإن `sort` يستخدم مفتاح يبدأ من الحقل المحدد وينتهي بآخر السطر. هذا هو شكل استخدام الترتيب بالاعتماد على أكثر من حقل:

```
[me@linuxbox ~]$ sort --key=1,1 --key=2n distros.txt
Fedora          5       03/20/2006
Fedora          6       10/24/2006
Fedora          7       05/31/2007
Fedora          8       11/08/2007
Fedora          9       05/13/2008
Fedora          10      11/25/2008
SUSE            10.1    05/11/2006
SUSE            10.2    12/07/2006
SUSE            10.3    10/04/2007
SUSE            11.0    06/19/2008
Ubuntu          6.06    06/01/2006
Ubuntu          6.10    10/26/2006
Ubuntu          7.04    04/19/2007
Ubuntu          7.10    10/18/2007
Ubuntu          8.04    04/24/2008
Ubuntu          8.10    10/30/2008
```

استخدمنا الصيغة الطويلة من الخيار للتوضيح، يمكن استخدام `-k 1,1 -k 2n` عوضًا عن الصيغة السابقة. في أول استخدام للخيار k-، حددنا مجالًا لتضمينه في المفتاح. ولأننا نريد أن نرتب أول حقل فقط، فنستخدم التعبير "1,1" أي البدء في الحقل الأول والانتهاء في الحقل الأول (تذكر أنه في حال عدم تحديد المجال بهذه الطريقة، فسيبدأ `sort` من الحقل الأول وسينتهي في آخر السطر). في الاستخدام الثاني للخيار k-، حددنا 2n أي استخدام الحقل الثاني كمفتاح للترتيب والقيام بالترتيب العددي. يمكن إضافة حرف في

آخر قيمة الخيار k- لتحديد نوع الترتيب الذي سيُنفّذ. يمكن أن يكون الحرف (كما في الخيارات العامة) "B" أي تجاهل الفراغات في أول الحقل، أو "n" أي الترتيب العددي، أو "r" أي الترتيب العكسي، وهكذا.

يحتوي الحقل الثالث في القائمة على التاريخ لكن بصيغة غير ملائمة للترتيب. تُنسَّق التواريخ في الحواسيب على الشكل: YYYY-MM-DD لتسهيل عملية الترتيب الزمني، لكن الصيغة التي استخدمناها هي الصيغة الأميركية MM/DD/YYYY. لذا، كيف نستطيع ترتيب القائمة زمنيًا؟

لحسن الحظ، يوفر sort طريقة لذلك. يسمح الخيار k- بتحديد الإزاحة (offset) داخل الحقول:

```
[me@linuxbox ~]$ sort -k 3.7nbr -k 3.1nbr -k 3.4nbr distros.txt
Fedora          10        11/25/2008
Ubuntu          8.10      10/30/2008
SUSE            11.0      06/19/2008
Fedora          9         05/13/2008
Ubuntu          8.04      04/24/2008
Fedora          8         11/08/2007
Ubuntu          7.10      10/18/2007
SUSE            10.3      10/04/2007
Fedora          7         05/31/2007
Ubuntu          7.04      04/19/2007
SUSE            10.2      12/07/2006
Ubuntu          6.10      10/26/2006
Fedora          6         10/24/2006
Ubuntu          6.06      06/01/2006
SUSE            10.1      05/11/2006
Fedora          5         03/20/2006
```

باستخدام 3.7 k- فإننا وجهنا sort لاستخدام مفتاح الترتيب الذي يبدأ عند المحرف السابع من الحقل الثالث، الذي يحدد بداية السنة. وبشكل مشابه، استخدمنا 3.1 k- و 3.4 k- لعزل الشهر واليوم من حقل التاريخ. أضفنا أيضًا الخيارَين n و r للحصول على ترتيب عددي معكوس. أُستخدم الخيار b لتجاهل المسافات (لأن الأرقام تختلف من سطرٍ إلى آخر) في حقل التاريخ. بعض الملفات لا تستخدم مسافات الجدولة والفراغات كفواصل؛ على سبيل المثال، ملف /etc/passwd :

```
[me@linuxbox ~]$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
```

```
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
```

تُفصَل الحقول في هذا الملف بنقطتَين رأسيتَين ":"، إذًا، كيف سنستطيع ترتيب هذا الملف باستخدام حقل مفتاحي؟ يوفر sort الخيار t- لتحديد المحرف الذي يفصل ما بين الحقول. لترتيب ملف passwd بالاعتماد على الحقل السابع (الصدفة الافتراضية)، بإمكاننا تنفيذ الآتي:

```
[me@linuxbox ~]$ sort -t ':' -k 7 /etc/passwd | head
me:x:1001:1001:Myself,,,:/home/me:/bin/bash
root:x:0:0:root:/root:/bin/bash
dhcp:x:101:102::/nonexistent:/bin/false
gdm:x:106:114:Gnome Display Manager:/var/lib/gdm:/bin/false
hplip:x:104:7:HPLIP system user,,,:/var/run/hplip:/bin/false
klog:x:103:104::/home/klog:/bin/false
messagebus:x:108:119::/var/run/dbus:/bin/false
polkituser:x:110:122:PolicyKit,,,:/var/run/PolicyKit:/bin/false
pulse:x:107:116:PulseAudio daemon,,,:/var/run/pulse:/bin/false
```

استطعنا الترتيب وفق الحقل السابع بجعل النقطتين الرأسيتين فاصلًا.

uniq

يُعتبَر البرنامج uniq "خفيفًا" مقارنةً مع sort. يقوم uniq بمهمة بسيطة للغاية. عندما يُمرَّر إليه ملف مرتب (يمكن استخدام مجرى الدخل القياسي أيضًا)، فسيحذف الأسطر المكررة ويُرسِل الناتج إلى مجرى الخرج القياسي. يُستخدم هذا الأمر عادةً مع sort لكي يزيل التكرارات من الناتج.

تلميحة: على الرغم من أن uniq هي أداة تقليدية في يونكس وتُستخدم عادةً مع sort، إلا أن نسخة غنو من sort تدعم الخيار u-، الذي يزيل التكرارات من الناتج.

لننشئ ملفًا لنجرب عليه الأمر uniq:

```
[me@linuxbox ~]$ cat > foo.txt
a
b
c
a
b
c
```

تذكر أن تضغط على `Ctrl-d` لإنهاء المدخلات التي ستُرسَل إلى الأمر من مجرى الدخل القياسي. يمكننا الآن تجربة الأمر `uniq` على الملف النصي السابق:

```
[me@linuxbox ~]$ uniq foo.txt
a
b
c
a
b
c
```

النتائج على حالها! لا يوجد أي اختلاف فيما بينها وما بين الملف الأصلي ولم تُحذف التكرارات! يجب أن يكون الملف مرتبًا لكي يقوم `uniq` بعمله:

```
[me@linuxbox ~]$ sort foo.txt | uniq
a
b
c
```

هذا لأن `uniq` يحذف الأسطر المتكررة المتوالية مع بعضها البعض.

لدى `uniq` العديد من الخيارات. وهذه أشهرها:

الجدول 20-2: خيارات uniq الشائعة

الخيار	الشرح
`-c`	طباعة قائمة بالأسطر المتكررة يتبعها عدد تكرارات تلك الأسطر.
`-d`	طباعة الأسطر المتكررة فقط.

-f n	تجاهل n حقلًا في بداية كل سطر. يُفصَل بين الحقول بفراغ أو مسافة جدولة، لكن لاحظ أن الأمر uniq لا يوفر ميزة تحديد الفاصل كما في sort.
-i	تجاهل حالة الأحرف عند القيام بالمقارنات بين الأسطر.
-s n	تجاهل أول n محرفًا من كل سطر.
-u	طباعة الأسطر الفريدة فقط. وهذا هو الخيار الافتراضي.

سنستخدم uniq في المثال الآتي للتبليغ عن عدد التكرارات الموجودة في ملفنا النصي وذلك باستخدام الخيار "c-":

```
[me@linuxbox ~]$ sort foo.txt | uniq -c
  2 a
  2 b
  2 c
```

التفريق والتجميع

تُستخدم البرامج الثلاثة التي سنناقشها الآن لفصل حقول نصية من ملف وإعادة تجميعها بشكل مفيد.

cut

يستخدم البرنامج cut لاستخراج قسم من النص في السطر وطباعته إلى مجرى الخرج القياسي. يقبل cut المدخلات على شكل ملفات تُمرَّر كوسائط أو عن طريق مجرى الدخل القياسي.

تحديد القسم الذي سيستخرج من السطر هو عملية معقدة بعض الشيء وتحدد باستخدام الخيارات الآتية:

الجدول 20-3: خيارات cut

الخيار	الشرح
-c char_list	استخراج قسم من السطر المُعرَّف بواسطة char_list. قد تحتوي القائمة على مجال عددي أو أكثر مفصولين بفاصلة ",".
-f field_list	استخراج حقل واحد أو أكثر معرفين بواسطة field_list من السطر. قد تحتوي القائمة على حقل واحد أو أكثر، أو على مجالات حقول مفصولين بفاصلة ",".
-d delim_char	استخدم delim_char رمزًا فاصلًا ما بين الحقول عند تحديد الخيار f-. يكون

	الفاصل ما بين الحقول افتراضيًّا هو مسافة الجدولة .
`--complement`	استخراج كامل السطر عدا الأجزاء المُحددة بواسطة c- و/أو f-.

كما لاحظنا، الطريقة التي يستخرج cut النص فيها هي طريقة غير مرنة. يستخدم cut استخدامًا فعّالًا لاستخراج النص من نواتج البرامج وليس من الملفات التي كتبها المستخدم بنفسه. سنلقي نظرة على ملف distros.txt ونحدد فيما إذا كان صالحًا للقيام بتجاربنا عليه باستخدام الأمر cut. إذا استخدمنا cat مع الخيار A-، فبإمكاننا معرفة إذا كانت حقول الملف مفصولة باستخدام مسافة الجدولة:

```
[me@linuxbox ~]$ cat -A distros.txt
SUSE^I10.2^I12/07/2006$
Fedora^I10^I11/25/2008$
SUSE^I11.0^I06/19/2008$
Ubuntu^I8.04^I04/24/2008$
Fedora^I8^I11/08/2007$
SUSE^I10.3^I10/04/2007$
Ubuntu^I6.10^I10/26/2006$
Fedora^I7^I05/31/2007$
Ubuntu^I7.10^I10/18/2007$
Ubuntu^I7.04^I04/19/2007$
SUSE^I10.1^I05/11/2006$
Fedora^I6^I10/24/2006$
Fedora^I9^I05/13/2008$
Ubuntu^I6.06^I06/01/2006$
Ubuntu^I8.10^I10/30/2008$
Fedora^I5^I03/20/2006$
```

يبدو أنه جيد. لا توجد فراغات، مجرد مسافة جدولة بين الحقول. ولأن الملف يستخدم مفاتيح الجدولة بدل الفراغات، فسوف نستخدم الخيار f- مباشرةً لاستخراج حقلٍ منه:

```
[me@linuxbox ~]$ cut -f 3 distros.txt
12/07/2006
11/25/2008
06/19/2008
04/24/2008
11/08/2007
```

```
10/04/2007
10/26/2006
05/31/2007
10/18/2007
04/19/2007
05/11/2006
10/24/2006
05/13/2008
06/01/2006
10/30/2008
03/20/2006
```

لما كان الفاصل في ملف `distros` هو مسافة الجدولة، فمن الأفضل استخدام `cut` لاستخراج الحقول وليس المحارف. لأنه عندما تُفصَل الحقول في ملف باستخدام مسافة الجدولة، فمن الصعب جدًا أو المستحيل حساب موضع حرفٍ ما في السطر. لكن لحسن الحظ، استخرجنا في المثال السابق بيانات بنفس الطول. لذا، بإمكاننا معرفة طريقة استخراج المحارف باستخراج السنة من كل سطر:

```
[me@linuxbox ~]$ cut -f 3 distros.txt | cut -c 7-10
2006
2008
2008
2008
2007
2007
2006
2007
2007
2007
2006
2006
2008
2006
2008
2006
```

بتنفيذ `cut` مرّة ثانية على قائمتنا، استطعنا استخراج المحارف ذات الموضع من 7 إلى 10 التي تُمثل السنة

في حقل التاريخ. "7-10" هو مثال عن تحديد مجال. تحتوي صفحة الدليل للأمر cut على شرح كامل لطريقة عمل المجالات.

نشر مفاتيح الجدولة

الملف `distros.txt` مُنسَّق تنسيقًا ممتازًا للسماح للأمر cut باستخراج الحقول منه. لكن ماذا لو أردنا معالجة الملف باستخدام cut بتحديد المحارف بدلًا من الحقول؟ يتطلب ذاك الأمر منا أن ننشر (expand) مفاتيح الجدولة ونحولها إلى العدد المناسب من الفراغات. لحسن الحظ، تحتوي حزمة `GNU Coreutils` على أداة لفعل ذلك. اسم تلك الأداة هو expand، يقبل هذا البرنامج المدخلات من مجرى الدخل القياسي، أو ملفٍ واحدٍ أو أكثر، ويُرسِل مخرجاته إلى مجرى الخرج القياسي.

إذا عالجنا ملف `distros.txt` باستخدام expand، فسيصبح باستطاعتنا استخدام `cut -c` لاستخراج أي مجال من المحارف في الملف. على سبيل المثال، نستطيع استخدام الأمر الآتي لاستخراج السنة من قائمتنا. وذلك بنشر مفاتيح الجدولة في الملف واستخدام cut لاستخراج كل المحارف الموجودة من الموضع 23 إلى نهاية السطر:

```
[me@linuxbox ~]$ expand distros.txt | cut -c 23-
```

توفر حزمة `Coreutils` أيضًا البرنامج unexpand لتحويل الفراغات إلى مفاتيح الجدولة.

عند التعامل مع الحقول، باستطاعتنا تحديد فاصل آخر للحقول عدا مفاتيح الجدولة. سنستخرج هنا أول حقل من ملف `/etc/passwd`:

```
[me@linuxbox ~]$ cut -d ':' -f 1 /etc/passwd | head
root
daemon
bin
sys
sync
games
man
lp
mail
news
```

غيرنا الفاصل إلى النقطتين الرأسيتين باستخدام الخيار d-.

paste

يقوم الأمر paste بعكس ما يقوم به cut. فبدلًا من استخراج حقل نصي من ملف، فإن paste يضيف حقلًا واحدًا أو أكثر إلى ملف. يقوم بذلك بقراءة أكثر من ملف وجمع الحقول الموجودة في كل ملف إلى مجرى موحد هو مجرى الخرج القياسي. وكما في cut، يقبل paste المدخلات من مجرى الخرج القياسي و/أو الملفات الممررة كوسائط. لشرح كيف يعمل paste، سنعالج الملف distros.txt لإنشاء قائمة مرتبة زمنيًا بإصدارات التوزيعات.

من تعاملنا السابق مع sort، سننشئ قائمة مرتبة زمنيًا بإصدارات التوزيعات ونخزنها في ملف باسم distros-by-date.txt:

```
[me@linuxbox ~]$ sort -k 3.7nbr -k 3.1nbr -k 3.4nbr distros.txt > dis-
tros-by-date.txt
```

سنستخدم الآن cut لاستخراج أول حقلين من الملف (اسم التوزيعة والإصدار)، وتخزين الناتج في ملف باسم distro-versions.txt:

```
[me@linuxbox ~]$ cut -f 1,2 distros-by-date.txt > distros-versions.txt
[me@linuxbox ~]$ head distros-versions.txt
Fedora          10
Ubuntu          8.10
SUSE            11.0
Fedora          9
Ubuntu          8.04
Fedora          8
Ubuntu          7.10
SUSE            10.3
Fedora          7
Ubuntu          7.04
```

آخر جزء من التحضيرات هو استخراج تواريخ إصدار التوزيعات وتخزينهم في ملف باسم distros-dates.txt:

```
[me@linuxbox ~]$ cut -f 3 distros-by-date.txt > distros-dates.txt
[me@linuxbox ~]$ head distros-dates.txt
11/25/2008
10/30/2008
```

```
06/19/2008
05/13/2008
04/24/2008
11/08/2007
10/18/2007
10/04/2007
05/31/2007
04/19/2007
```

لدينا الآن جميع الأقسام التي نريدها لإكمال عملية المعالجة. سنستخدم paste لوضع حقل التواريخ قبل أسماء التوزيعات وأرقام إصداراتهم، وهذا ما يُنشئ قائمةً مرتبةً زمنيًا. يمكن القيام بذلك بتحديد الترتيب الصحيح لوسائط الأمر paste كالآتي:

```
[me@linuxbox ~]$ paste distros-dates.txt distros-versions.txt
11/25/2008      Fedora      10
10/30/2008      Ubuntu      8.10
06/19/2008      SUSE        11.0
05/13/2008      Fedora      9
04/24/2008      Ubuntu      8.04
11/08/2007      Fedora      8
10/18/2007      Ubuntu      7.10
10/04/2007      SUSE        10.3
05/31/2007      Fedora      7
04/19/2007      Ubuntu      7.04
12/07/2006      SUSE        10.2
10/26/2006      Ubuntu      6.10
10/24/2006      Fedora      6
06/01/2006      Ubuntu      6.06
05/11/2006      SUSE        10.1
03/20/2006      Fedora      5
```

join

يمكن اعتبار الأمر join مشابهًا للأمر paste حيث أنه يسمح بإضافة الحقول إلى ملف، لكنه يستخدم طريقته الخاصة (والفريدة) لفعل ذلك. عملية الضم (join) هي عملية تتم على قواعد البيانات العلائقية (relational databases) حيث تُجمَع البيانات القادمة من عدّة جداول (tables) تتشارك في حقل مفتاحي (key

field) في نتيجة واحدة. يعمل برنامج `join` بنفس تلك الآلية. إنه يجمع البيانات من عدّة ملفات بالاعتماد على حقل مفتاحي معين.

لكي نرى كيف تُنفَّذ عملية الضم في قواعد البيانات العلائقية، فعلينا أن نتخيل قاعدة بيانات صغيرة للغاية تحتوي على جدولين كلٌّ منهما يحتوي على سجل (record) واحد. الجدول الأول المسمى CUSTOMERS يحتوي على ثلاثة حقول: رقم الزبون (CUSTNUM)، والاسم الأول للزبون (FNAME)، والاسم الأخير للزبون (LNAME):

```
CUSTNUM        FNAME        LNAME
========       =====        ======
4681934        John         Smith
```

الجدول الثاني يدعى ORDERS، ويحتوي على أربعة حقول: رقم الطلبية (ORDERNUM)، ورقم الزبون (CUSTNUM)، والكميّة المطلوبة (QUAN)، والمنتج المطلوب (ITEM).

```
ORDERNUM        CUSTNUM        QUAN        ITEM
========        =======        ====        ====
3014953305      4681934        1           Blue Widget
```

لاحظ أن كلا الجدولين يحتوي على حقل مشترك CUSTNUM. وهذا ما يسمح بإنشاء علاقة ما بين الجدولين.

تسمح عملية الضم لنا بجمع الحقول في كلا الجدولين للحصول على نتيجة مفيدة، كالتحضير للفاتورة على سبيل المثال. باستخدام القيم المتطابقة للحقل CUSTNUM في كلا الجدولين؛ ستُنتِج عملية الضم المخرجات الآتية:

```
FNAME          LNAME          QUAN        ITEM
=====          =====          ====        ====
John           Smith          1           Blue Widget
```

سنحتاج إلى إنشاء ملفَين يوجد بينهما مفتاح مشترك لكي نشرح عمل الأمر `join`. سنستخدم ملف `distros-by-date.txt` السابق. سننشئ ملفَين إضافيَين من هذا الملف، يحتوي أحدهما على تاريخ الإصدار (الذي سيكون الحقل المشترك) واسم التوزيعة:

```
[me@linuxbox ~]$ cut -f 1,1 distros-by-date.txt > distros-names.txt
[me@linuxbox ~]$ paste distros-dates.txt distros-names.txt > distros-key-names.txt
```

```
[me@linuxbox ~]$ head distros-key-names.txt
11/25/2008 Fedora
10/30/2008 Ubuntu
06/19/2008 SUSE
05/13/2008 Fedora
04/24/2008 Ubuntu
11/08/2007 Fedora
10/18/2007 Ubuntu
10/04/2007 SUSE
05/31/2007 Fedora
04/19/2007 Ubuntu
```

والملف الثاني الذي يحتوي على تاريخ الإصدار ورقم نسخة التوزيعة:

```
[me@linuxbox ~]$ cut -f 2,2 distros-by-date.txt > distros-vernums.txt
[me@linuxbox ~]$ paste distros-dates.txt distros-vernums.txt > distros-key-vernums.txt
[me@linuxbox ~]$ head distros-key-vernums.txt
11/25/2008 10
10/30/2008 8.10
06/19/2008 11.0
05/13/2008 9
04/24/2008 8.04
11/08/2007 8
10/18/2007 7.10
10/04/2007 10.3
05/31/2007 7
04/19/2007 7.04
```

أصبح لدينا الآن ملفَين بينهما مفتاح مشترك (حقل "تاريخ الإصدار"). يجدر بالذكر أنه يجب أن تكون الملفات مرتبةً كي يعمل الأمر join بنجاح:

```
[me@linuxbox ~]$ join distros-key-names.txt distros-key-vernums.txt | head
11/25/2008 Fedora 10
10/30/2008 Ubuntu 8.10
```

```
06/19/2008 SUSE 11.0
05/13/2008 Fedora 9
04/24/2008 Ubuntu 8.04
11/08/2007 Fedora 8
10/18/2007 Ubuntu 7.10
10/04/2007 SUSE 10.3
05/31/2007 Fedora 7
04/19/2007 Ubuntu 7.04
```

لاحظ أيضًا أن `join` يَستخدم (افتراضيًا) الفراغات ومسافات الجدولة كفاصل ما بين الحقول، ويُخرِج فراغًا واحدًا ما بين الحقول في المخرجات. يمكن تعديل هذا السلوك بتحديد بعض الخيارات. راجع صفحة الدليل للأمر `join` للمزيد من التفاصيل.

مقارنة النصوص

من المفيد مقارنة نِسخ مختلفة من الملفات النصية. وهذا أمر محوري ومهم خصوصًا لمدراء الأنظمة ومطوري البرمجيات. على سبيل المثال، يحتاج مدير النظام إلى مقارنة أحد ملفات الإعدادات الحالية بنسخة قديمة منه كي يُشخِّص أو يُحلّل مشكلةً ما في النظام. وبشكلٍ مشابه، يطّلع المبرمج على التعديلات التي أُجريت على الكود المصدري للتطبيقات من حينٍ لآخر.

comm

يقارن البرنامج comm ملفَين نصيَين ويظهر الأسطر الفريدة والأسطر المشتركة في كل ملف. لكي نشرح عمله، سنُنشئ ملفَين نصيَين متشابهَين باستخدام `cat`:

```
[me@linuxbox ~]$ cat > file1.txt
a
b
c
d
[me@linuxbox ~]$ cat > file2.txt
b
c
d
e
```

سنقارن الآن الملفات باستخدام comm:

```
[me@linuxbox ~]$ comm file1.txt file2.txt
a
                b
                c
                d
       e
```

كما لاحظنا، يُظهِر comm ثلاثة حقول من المخرجات. يحتوي الحقل الأول على الأسطر التي انفرد بها الملف الأول؛ يحتوي الحقل الثاني على الأسطر التي انفرد بها الملف الثاني؛ أما الحقل الثالث، فيحتوي على الأسطر المشتركة ما بين الملفين. يدعم comm الخيار n- حيث تأخذ n القيم 1 أو 2 أو 3. يُحدِّد هذا الخيار أيّ حقل أو حقول لا يجب عرضها. على سبيل المثال، لو أردنا أن نُظهِر الأسطر المشتركة ما بين الملفين فقط، فإننا نستخدم الخيار n- مع رقمي الحقلين 1 و 2:

```
[me@linuxbox ~]$ comm -12 file1.txt file2.txt
b
c
d
```

diff

يُستخدم `diff`، كما في برنامج comm، لاكتشاف الاختلافات ما بين الملفات. لكن `diff` هو أداة معقدة جدًا تدعم العديد من تنسيقات المخرجات وقادرة على معالجة مجموعات ضخمة من الملفات في آن واحد. يُستخدم `diff` عادةً من مطوري البرامج لتفحص التغيرات بين إصدارات مختلفة من الكود المصدري للبرامج، ويمكن استخدام `diff` لتفحص جميع الملفات الموجودة ضمن مجلدٍ ما (وجميع المجلدات الفرعية الموجودة فيه) المعروفة بمصطلح "شجرة المصدر" (source tree). أحد أشهر استخدامات `diff` هي إنشاء ملف الاختلافات (`diff`) أو ما يسمى الرُقع (`patches`) التي يمكن استخدامها مع برامج مثل `patch` (الذي سنناقشه بعد قليل) لتحويل إصدارة أحد الملفات إلى إصدارٍ آخر.

إذا طبقنا الأمر `diff` على الملفَين اللذَين استخدمناهما في المثال السابق:

```
[me@linuxbox ~]$ diff file1.txt file2.txt
1d0
< a
4a4
```

```
> e
```

سنشاهد النمط الافتراضي للمخرجات: شرح موجز عن الفروقات ما بين الملفين. في النمط الافتراضي، كل مجموعة من التغييرات مسبوقة بأمر التغيير (change command) الذي يكون على الشكل "المجال العملية المجال"، لوصف مواضع وأنواع التغيرات المطلوبة لتحويل الملف الأول إلى الملف الثاني.

الجدول 20-4: أوامر التغيير في diff

الخيار	الشرح
r1ar2	إضافة الأسطر الموجودة في الموضع r2 في الملف الثاني إلى الموضع r1 في الملف الأول.
r1cr2	استبدال الأسطر في الموضع r1 بالأسطر الموجودة في الموضع r2 في الملف الثاني.
r1dr2	حذف الأسطر الموجود في الملف الأول في الموضع r1، التي ظهرت في الموضع r2 في الملف الثاني.

في التنسيق أو النمط الافتراضي لبرنامج `diff`، يمكن أن يحتوي المجال على رقمين مفصولين بفاصلة للإشارة إلى رقم سطر بداية المجال ورقم سطر نهاية المجال (وذلك للالتزام بمعايير POSIX وللتوافق مع النسخ الأُخرى من `diff` التي يستخدمها يونكس)، هذا التنسيق غير مشهور كما باقي التنسيقات الاختيارية. أشهر تنسيقين هما "context format" و "unified format".

عند عرض الاختلافات بصيغة context format (باستخدام الخيار `c-`)، سنشاهد المخرجات الآتية:

```
[me@linuxbox ~]$ diff -c file1.txt file2.txt
*** file1.txt  2008-12-23 06:40:13.000000000 -0500
--- file2.txt  2008-12-23 06:40:34.000000000 -0500
***************
*** 1,4 ****
- a
  b
  c
  d
--- 1,4 ----
  b
  c
  d
+ e
```

تبدأ المخرجات باسمَيّ الملفَين وبصمة الوقت الخاصة بهما. أول ملف يكون مُعلّمًا بواسطة رمز "*" والملف الثاني بواسطة الشرطة. سيشير هذان الرمزان إلى ملفيهما الأصليَين. سنشاهد الآن مجموعات التغييرات، التي تحتوي على عدد الأسطر. تبدأ أول مجموعة بالآتي:

```
*** 1,4 ***
```

التي تعني: الأسطر من واحد إلى أربعة في الملف الأول. بعد ذلك نجد:

```
--- 1,4 ---
```

التي تعني: الأسطر من واحد إلى أربعة في الملف الثاني. يمكن أن تبدأ الأسطر داخل المجموعة بأحد الرموز الأربعة الآتية:

الجدول 20-5: رموز التغيير لنمط context

الرمز	الشرح
لا شيء	لا يوجد أي فرق ما بين الملفين في هذا السطر.
-	حُذِف السطر. أي أن السطر موجود في الملف الأول وليس موجودًا في الملف الثاني.
+	أُضيف سطر. أي أن السطر موجود في الملف الثاني وغير موجود في الملف الأول.
!	غُيّر السطر. ستُظهر نسختي السطر، كلٌ في مجموعته.

أما عند استخدام unified format (الذي يُحَدَّد بالخيار u-)، فسيكون الناتج موجزًا ومختصرًا كالآتي:

```
[me@linuxbox ~]$ diff -u file1.txt file2.txt
--- file1.txt  2008-12-23 06:40:13.000000000 -0500
+++ file2.txt  2008-12-23 06:40:34.000000000 -0500
@@ -1,4 +1,4 @@
-a
 b
 c
 d
+e
```

أكثر فرق ظاهر للعيان بين نمطي context و unified هو إزالة الأسطر المكررة من الناتج، وهذا ما يجعل المخرجات في نمط unified أقصر من نظيرتها في context. أُظهِرَت بصمات الوقت في أعلى الناتج تتبعها السلسلة النصية "@@ 1,4+ 1,4- @@" التي تشير إلى مجال الأسطر في الملف الأول، ومجال الأسطر في

الملف الثاني. ومن ثم الأسطر أنفسها؛ التي قد تُسبق بأحد الرموز الآتية:

الجدول 20-6: رموز التغيير لنمط unified

الرمز	الشرح
لا شيء	هذا السطر مشترك ما بين الملفين.
-	حُذِف هذا السطر من الملف الأول.
+	أُضيف هذا السطر إلى الملف الأول.

patch

يُستخدم البرنامج patch لتطبيق التغيرات إلى الملفات النصية. يقبل patch مخرجات diff كمدخلات له؛ ويُستخدم عادةً لتحويل إصدار قديم من الملفات إلى إصدارٍ أحدث. لنفترض المثال الآتي: تُطَوَّر نواة لينُكس من العديد من فرق المتطوعين المنتشرين حول العالم الذين يُحدِثون تغييرات بسيطة على الكود المصدري للنواة. تحتوي نواة لينُكس على ملايين الأسطر البرمجية؛ بينما تكون التغييرات التي يقوم بها المتطوع الواحد في كل مرّة صغيرة جدًا. فليس من المعقول أن يُرسِل المتطوع جميع الأكواد المصدرية الخاصة بالنواة لجميع المطورين في كل مرة يُحدِث فيها تعديلًا بسيطًا؛ إلا أنه يرسل عوضًا عن ذلك ملف الفروقات diff. يحتوي ملف diff على جميع التغيرات التي حصلت بين النسختَين القديمة والحديثة (أي التي عدلها المتطوع) من الملف. يستخدم بعد ذلك المتلقي برنامج patch لتطبيق التغيرات على الملفات المصدرية التي لديه. يقدّم استخدام diff/patch ميزتَين أساسيتَين:

1. حجم ملف diff صغير جدًا بالمقارنة مع حجم كامل الكود المصدري.
2. يُظهِر ملف diff التغيرات التي حدثت على الملف بوضوح؛ مما يُمكّن مراجعي الكود من فهم ما الذي عُدِّل بسرعة ويسر.

يمكن تطبيق diff/patch على أي ملف نصي، وليس فقط على الأكواد البرمجية. سيكون استخدامه عمليًا على ملفات الإعدادات وغيرها.

ينصح توثيق غنو باستخدام diff على الشكل الآتي لتحضير ملف diff للاستخدام مع patch:

```
diff -Naur old_file new_file > diff_file
```

حيث old_file و new_file هما ملفان نصيان أو مجلدان يحتويان على الملفات.

بعد أن يُنشأ ملف الاختلافات diff، نستطيع الآن أن "نُرقِّع" الملف القديم ونحوّل محتوياته إلى نفس محتويات الملف الجديد:

```
patch < diff_file
```

لنجرب ذلك مع ملفاتنا السابقة:

```
[me@linuxbox ~]$ diff -Naur file1.txt file2.txt > patchfile.txt
[me@linuxbox ~]$ patch < patchfile.txt
patching file file1.txt
[me@linuxbox ~]$ cat file1.txt
b
c
d
e
```

أنشأنا في المثال السابق ملف فروقات باسم `patchfile.txt` ومن ثم استخدمنا برنامج `patch` لتطبيق الرقعة. لاحظ أننا لم نُحدِّد الملف الهدف للأمر `patch`، لأن ملف الفروقات (بتنسيق `unified`) يحتوي على أسماء الملفات في ترويسته. بعد أن طُبِّقَت الرقعة، سنجد أن محتويات الملف `file1.txt` أصبحت تُطابِق محتويات الملف `file2.txt`.

لدى برنامج `patch` عدد كبير من الخيارات، ويوجد أيضًا عدد من البرامج التي تستطيع أن تحلل وتعدل ملفات الرقع.

تعديل النصوص بطرق غير تفاعلية

تجاربنا السابقة مع التعديل كانت "تفاعلية"، أي أننا كنا نحرّك المؤشر تحريكًا يدويًا وندخل التعديلات... لكن يوجد هنالك طرق "غير تفاعلية" لتعديل النصوص. تسمح هذه الطرق أيضًا بأن تُعدَّل عدّة ملفات سويةً باستخدام أمر واحد فقط.

tr

يُستخدم برنامج `tr` "لتحويل" المحارف، يمكننا تخيل عملية التحويل على أنها البحث عن محارف واستبدالها بأُخرى. على سبيل المثال: استبدال جميع الأحرف الصغيرة بالأحرف الكبيرة هو عملية "تحويل" (`transliteration`). يمكننا فعل ذلك باستخدام `tr` كالآتي:

```
[me@linuxbox ~]$ echo "lowercase letters" | tr a-z A-Z
LOWERCASE LETTERS
```

كما لاحظنا، يقبل `tr` المدخلات من مجرى الدخل القياسي ويُرسِل المخرجات إلى مجرى الخرج القياسي. يقبل

tr وسيطين: الأول هو مجموعة المحارف التي ستُحوَّل، والثاني هو مجموعة المحارف التي سيُحوَّل إليها. يمكن التعبير عن المحارف بإحدى الطرق الآتية:

1. قائمة بالمحارف. مثال: ABCDEFGHIJKLMNOPQRSTUVWXYZ.
2. استخدام المجالات. على سبيل المثال A-Z. لكن انتبه أن هذه الطريقة قد تخضع لنفس المشاكل التي واجهتنا في الأوامر الأخرى بسبب ترتيب المحارف المستخدم في النظام. لذا يجب أخذ الحيطة والحذر عند استخدام المجالات.
3. فئات محارف POSIX. على سبيل المثال [:upper:].

يجب أن يكون (في أغلب الحالات) طول كلا الوسيطين متساويًا؛ لكن من الممكن أن تكون المجموعة الأولى أكبر من المجموعة الثانية، وخصوصًا إذا أردنا استبدال عدّة محارف بمحرف واحد فقط:

```
[me@linuxbox ~]$ echo "lowercase letters" | tr [:lower:] A
AAAAAAAAA AAAAAAA
```

يسمح tr (بالإضافة إلى التحويل) بأن تُزال بعض المحارف القادمة من مجرى الدخل. ناقشنا مشكلة تحويل الملفات النصية من صيغة DOS إلى صيغة يونكس في أول هذا الفصل. للقيام بهذا التحويل، يجب أن تحذف جميع محارف العودة إلى بداية السطر الموجودة في الملف. والتي يمكن القيام بها في tr كالآتي:

```
tr -d '\r' < dos_file > unix_file
```

حيث dos_file هو الملف الذي سيحوَّل ويخزَّن في ملف unix_file. يَستخدم الأمر السابق الرمز "\r" للإشارة إلى محرف العودة إلى بداية السطر. يمكن تنفيذ الأمر الآتي للحصول على قائمة كاملة بجميع المحارف أو الرموز أو الفئات التي يمكن استخدامها مع tr:

```
[me@linuxbox ~]$ tr --help
```

ROT13: حلقة فك الشيفرة "غير السرية"

أحد الاستخدامات المسلية للبرنامج tr هو تشفير النص بطريقة ROT13، وهي طريقة تشفير "سخيفة" مبنية على استبدال المحارف بمحارف أُخرى. إطلاق كلمة "تشفير" على ROT13 هو كرم كبير؛ مصطلح "تشويش النص" هو مصطلح أدق لوصفها. تستبدل هذه الخوارزمية كل حرف بالحرف الذي أمامه بثلاثة عشر حرفًا، ولأن 13 هو نصف عدد حروف اللغة الإنكليزية البالغ عددها 26 حرفًا؛ فإن

تنفيذ الخوارزمية مرّة أُخرى سيُعيد النص إلى حالته الأصلية. لتنفيذ تلك الخوارزمية باستخدام tr:

```
echo "secret text" | tr a-zA-Z n-za-mN-ZA-M
frperg grkg
```

تؤدي إعادة تنفيذ الأمر السابق على الناتج إلى استعادة النص الأصلي:

```
echo "frperg grkg" | tr a-zA-Z n-za-mN-ZA-M
secret text
```

يدعم عدد من عملاء البريد الإلكتروني وأخبار USENET خوارزمية ROT13. تحتوي ويكيبيديا على مقالة جيدة عن هذا الموضوع:

http://en.wikipedia.org/wiki/ROT13

يمكن للأمر tr أن يقوم بخدعة ثانية: استخدام الخيار s- يجعل tr يحذف الأحرف المكررة:

```
[me@linuxbox ~]$ echo "aaabbbccc" | tr -s ab
abccc
```

تحتوي السلسلة النصية في المثال السابق على أحرف مكررة. بتمرير المجموعة "ab" إلى tr، حذفنا جميع التكرارات الزائدة للحرفين "a" و "b" بينما تركنا تكرارات الحرف "c". لاحظ أن تكرارات الحرف يجب أن تكون متلاصقة. وإذا لم تكن متلاصقة:

```
[me@linuxbox ~]$ echo "abcabcabc" | tr -s ab
abcabcabc
```

فلن يُحذَف أي شيء.

sed

جاء الاسم sed من الكلمتين stream editor أي محرر تدفقي. يقوم sed بعمليات تعديل النص سواءً على مجرى الدخل القياسي أو على الملفات. sed هو برنامج قوي جدًا ومعقد (توجد كتب كاملة تتحدث عنه!)، لذا، لن نستطيع شرح جميع ميزاته هنا.

الطريقة التي يعمل sed فيها عمومًا هي قبول تعليمة واحدة من سطر الأوامر أو اسم ملف نصي يحتوي على عدّة تعليمات (التعليمات التي نتحدث عنها هنا هي تعليمات برنامج sed)، ومن ثم ينفذ sed تلك التعليمات على كل سطر في الملف (أو مجرى الدخل). هذا مثال بسيط لاستخدام sed:

```
[me@linuxbox ~]$ echo "front" | sed 's/front/back/'
back
```

طبعنا، في المثال السابق، كلمةً واحدةً باستخدام echo وأرسلناها عبر الأنبوب إلى sed، الذي بدوره نفذ التعليمة `s/front/back/` على النص وأخرج الكلمة back نتيجةً لتلك التعليمة. قد نتذكر أن تلك التعليمة تُشبه تعليمة الاستبدال التي استخدمناها في `vi`.

تبدأ التعليمات في sed بحرف واحد. أُستخدِم في المثال أعلاه الحرف "s" الذي هو اختصار للكلمة "substitution" أي استبدال؛ يتبع ذاك الحرف عبارتَيّ البحث والاستبدال مفصولتَين بخط مائل "/". يمكنك اختيار أي محرف ليكون هو الفاصل، لكن أصبح عُرفًا أن يُستخدَم الخط المائل كمحرف فصل، لكن sed يعتبر أي محرف يلي التعليمة هو محرف فصل. يمكننا تنفيذ الأمر السابق بالطريقة الآتية:

```
[me@linuxbox ~]$ echo "front" | sed 's_front_back_'
back
```

استخدام الشرطة السفلية مباشرةً بعد التعليمة "s" جعلها فاصلًا بدلًا من الخط المائل. إمكانية تغيير الفاصل تُستخدم عند الضرورة لجعل قراءة التعليمات أكثر سهولةً كما سنرى لاحقًا.

يمكن أن تُسبَق أغلب تعليمات sed بعنوان (address)، الذي يُحدِّد السطر أو الأسطر التي ستُعدَّل من المدخلات. إذا لم يُصرَّح عن العنوان، فستُنفذ التعليمة على جميع أسطر الملف (أو المجرى). أبسط أنواع العناوين هو ذكر رقم السطر، كما في المثال الآتي:

```
[me@linuxbox ~]$ echo "front" | sed '1s/front/back/'
back
```

إضافة العنوان 1 إلى التعليمة جعل الاستبدال يجري على السطر الأول من المدخلات (التي هي بدورها سطر واحد فقط). إذا حددنا رقم سطر آخر:

```
[me@linuxbox ~]$ echo "front" | sed '2s/front/back/'
front
```

فلن يتم التعديل، لأن المدخلات لا تحتوي إلا على سطرٍ واحد.

يمكن تحديد العناوين بطرقٍ مختلفة، هذه أشهرها:

الجدول 20-7: طرق تحديد العناوين في sed

الخيار	الشرح
n	رقم السطر، حيث n هو عدد موجب.
$	آخر سطر من الملف
/regexp/	الأسطر التي تُطابِق نمط التعابير النظامية الأساسي الذي يتبع معيار POSIX. لاحظ أن التعبير النظامي محاط بخط مائل "/". يمكن أن يُغيَّر الفاصل في التعبير النظامي إلى محرفٍ آخر وذلك بتحديده بالشكل الآتي: \cregexpc حيث c هو محرف الفصل.
addr1,addr2	تحديد مجال من الأسطر من addr1 إلى addr2، (متضمنًا تلك الأسطر). يمكن أن يكون شكل العنوان addr1 أو addr2 أحد أشكال العناوين السابقة.
first~step	تحديد السطر المحدد بالرقم first ومن ثم كل سطر يتبعه بعدد step من الأسطر. على سبيل المثال 2~1 تعني كل سطر ترتيبه فردي، 5~5 يشير إلى السطر الخامس وكل سطر يتبعه بخمسة أسطر وهكذا.
addr1,+n	يطابق السطر ذا العنوان addr1 و n سطرًا بعده.
addr!	يطابق كل سطر عدا السطر addr، الذي يمكن أن يكون أحد الأشكال السابقة.

سنشرح مختلف أنواع العناوين باستخدام ملف distros.txt الذي أنشأناه سابقًا في هذا الفصل. سنجرب أولًا مجال الأسطر:

```
[me@linuxbox ~]$ sed -n '1,5p' distros.txt
SUSE          10.2  12/07/2006
Fedora        10    11/25/2008
SUSE          11.0  06/19/2008
Ubuntu        8.04  04/24/2008
Fedora        8     11/08/2007
```

طبعنا، في المثال السابق مجالًا للأسطر بدءًا من السطر الأول إلى السطر الخامس. استخدمنا التعليمة p للقيام بذلك، التي تطبع الأسطر المطابقة. لجعل هذه التعليمة مفيدة، استخدمنا الخيار n- الذي يجعل sed لا يطبع جميع الأسطر تلقائيًا.

سنجرب الآن تحديد العنوان على شكل تعبير نظامي:

```
[me@linuxbox ~]$ sed -n '/SUSE/p' distros.txt
SUSE         10.2  12/07/2006
SUSE         11.0  06/19/2008
SUSE         10.3  10/04/2007
SUSE         10.1  05/11/2006
```

تمكنا من عزل الأسطر التي تحتوي على الكلمة "SUSE" باستخدام التعبير النظامي (المفصول بالخط المائل) /SUSE/ بشكل مشابه للأمر grep.

أخيرًا، سنجرب استخدام "!" لعكس العنوان:

```
[me@linuxbox ~]$ sed -n '/SUSE/!p' distros.txt
Fedora       10    11/25/2008
Ubuntu       8.04  04/24/2008
Fedora       8     11/08/2007
Ubuntu       6.10  10/26/2006
Fedora       7     05/31/2007
Ubuntu       7.10  10/18/2007
Ubuntu       7.04  04/19/2007
Fedora       6     10/24/2006
Fedora       9     05/13/2008
Ubuntu       6.06  06/01/2006
Ubuntu       8.10  10/30/2008
Fedora       5     03/20/2006
```

أخرج المثال السابق القائمة التي قد توقعتها: جميع الأسطر التي لا تحتوي على التعبير النظامي /SUSE/.

ألقينا حتى الآن نظرة على تعليمتَين من تعليمات sed: تعليمة الاستبدال "s" وتعليمة الطباعة "p". هذه قائمة بالتعليمات الأساسية التي يمكن استخدامها مع sed:

الجدول 20-8: تعليمات التعديل الأساسية في sed

الخيار	الشرح
=	إخراج السطر الحالي.
a	إضافة النص بعد السطر المحدد.
d	حذف السطر الحالي.

i	إضافة نص قبل السطر المحدد.
p	طباعة السطر الحالي. افتراضيًا، يطبع sed جميع الأسطر ويُعدِّل بعض الأسطر حسب التعليمة الممررة إليه. يمكن أن يُغيِّر هذا السلوك باستخدام الخيار n-.
q	الخروج من sed دون معالجة أيّة أسطر إضافية. طباعة السطر الحالي إن لم يكن الخيار n- محددًا.
Q	الخروج من sed دون معالجه أيّة أسطر إضافية.
s/regexp/replacement/	استبدال جميع السلاسل النصية التي تطابق التعبير النظامي regexp بالسلسلة النصية replacement. يمكن أن تحتوي السلسلة النصية التي سيتم الاستبدال بها على رمز "&" الذي يُمثل النص الذي تمت مطابقته باستخدام regexp. بالإضافة إلى ذلك، فإن السلسلة النصية التي سيتم الاستبدال بها قد تحتوي على التعبيرات "1\" إلى "9\" التي تعني محتويات الأنماط الفرعية في التعبير النظامي regexp. لمزيد من المعلومات راجع فقرة "الأنماط الفرعية" في الأسفل. يمكن أن تُحدَّد بعض الخيارات بعد فاصل النهاية (الذي يكون عادةً الخط المائل) لتخصيص سلوك التعليمة s.
y/set1/set2	إبدال جميع المحارف الموجود في المجموعة set1 بمناظراتها الموجودة في المجموعة set2. لاحظ أن التعليمة y تتطلب أن تكون المجموعتان بنفس الطول، عكس الأمر tr.

تعليمة الاستبدال s هي أكثر تعليمات التعديل استخدامًا. سنشرح طريقة استخدامها بتجربة التعديلات على ملف distros.txt. ناقشنا سابقًا أن الملف distros.txt ليس مناسبًا للترتيب الزمني لأن صيغة الوقت الموجودة فيه هي MM/DD/YYYY لكن الصيغة التي يتطلبها الترتيب الزمني هي YYYY-MM-DD. للقيام بهذا التعديل على الملف يدويًا، فسوف نحتاج إلى وقتٍ كثير وقد تحدث أخطاء غير متوقعة؛ لكن باستخدام sed، نستطيع القيام بهذا التعديل بخطوة واحدة:

```
[me@linuxbox ~]$ sed 's/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)
$/\3-\1-\2/' distros.txt
SUSE         10.2    2006-12-07
```

```
Fedora       10      2008-11-25
SUSE         11.0    2008-06-19
Ubuntu       8.04    2008-04-24
Fedora       8       2007-11-08
SUSE         10.3    2007-10-04
Ubuntu       6.10    2006-10-26
Fedora       7       2007-05-31
Ubuntu       7.10    2007-10-18
Ubuntu       7.04    2007-04-19
SUSE         10.1    2006-05-11
Fedora       6       2006-10-24
Fedora       9       2008-05-13
Ubuntu       6.06    2006-06-01
Ubuntu       8.10    2008-10-30
Fedora       5       2006-03-20
```

جميل جدًا! باستخدام الأمر القبيح السابق، غيرنا صيغة التاريخ في الأمر السابق وبخطوة واحدة فقط! هذا مثال واضح عن سبب وصف التعابير النظامية بأنها "للكتابة فقط" :-) . يمكنك كتابة التعابير النظامية، لكن في بعض الأحيان لا تستطيع قراءتهم. قبل أن نهرب بعيدًا من الأمر المرعب السابق. لنلقِ نظرة على الطريقة التي بُني فيها. أولًا، نحن نعرف أن الأمر السابق تكون بنيته الأساسية على الشكل الآتي:

```
sed 's/regexp/replacement/' distros.txt
```

الخطوة التالية هي كتابة التعبير النظامي الذي سيطابق التاريخ. ولأنه من الشكل MM/DD/YYYY ويظهر في آخر السطر؛ فبإمكاننا كتابة نمط كالآتي:

```
[0-9]{2}/[0-9]{2}/[0-9]{4}$
```

الذي يطابق رقمين ومن ثم خط مائل، ثم رقمين ومن ثم خط مائل، وبعدها أربعة أرقام ونهاية السطر. حسنًا، وضّح الشرح السابق تعبير المطابقة regexp، لكن ماذا عن تعبير الاستبدال (replacement)؟ يجب علينا الآن أن نتعرف على ميزة جديدة في التعابير النظامية تسمى "التعابير الفرعية" أو الأنماط الفرعية: إذا ظهرت العبارة n\ (حيث n هي رقم من 1 إلى 9) في تعبير الاستبدال فإن تلك العبارة ستشير إلى النمط الفرعي المطابق لها في عبارة البحث regexp. لإنشاء أنماط فرعية، فإننا نضع التعابير بين أقواس كالآتي:

```
([0-9]{2})/([0-9]{2})/([0-9]{4})$
```

لدينا الآن ثلاثة تعابير فرعية. أول تعبير يحتوي على الشهر، والثاني على اليوم، والثالث على السنة. يمكننا الآن بناء تعبير الاستبدال كالآتي:

```
\3-\1-\2
```

الذي يعطينا: "السنة - الشهر - اليوم".

لذا ستكون تعليمتنا على الشكل الآتي:

```
sed 's/([0-9]{2})/([0-9]{2})/([0-9]{4})$/\3-\1-\2/' distros.txt
```

يبقى الآن لدينا مشكلتان فقط، الأولى هي أن الخطوط المائلة "/" في التعبير النظامي المستخدم ستُربِك sed وستجعله يظن أنه أنهى التعليمة s. الثانية هي أن sed يقبل افتراضيًّا التعابير النظامية الأساسية وليس الموسعة. مما يجعل بعض المحارف في التعبير السابق تُعامَل على أنها أحرف عادية بدل اعتبارها أحرفًا خاصة. تُحَلّ كلتا المشكلتين باستخدام الشرطة المائلة الخلفية "\" لتهريب تلك المحارف:

```
sed 's/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/'
distros.txt
```

ميزة أخرى من مزايا تعليمة الاستبدال s هي استخدام الرايات الاختيارية التي تلي عبارة الاستبدال. أهم تلك الرايات هي الراية g، التي تجعل sed يستبدل جميع المطابقات في السطر وليس أول مطابقة فقط. جرّب المثال الآتي:

```
[me@linuxbox ~]$ echo "aaabbbccc" | sed 's/b/B/'
aaaBbbccc
```

لاحظنا أن الاستبدال جرى لأول حرف "b" في السطر. نستطيع، باستخدام الراية g، استبدال جميع المطابقات:

```
[me@linuxbox ~]$ echo "aaabbbccc" | sed 's/b/B/g'
aaaBBBccc
```

حتى الآن، أعطينا جميع التعليمات إلى sed باستخدام سطر الأوامر؛ من الممكن تخزين التعليمات المعقدة في ملف منفصل واستدعاؤها بالخيار f-. سنستخدم sed الآن مع ملف distros.txt لإنشاء تقرير يحتوي على عنوان في الأعلى والتواريخ المعدلة إلى الشكل الجديد وأسماء التوزيعات مكتوبةً بأحرف كبيرة. الآن شغل محررك المفضل واكتب الآتي:

```
# sed script to produce Linux distributions report
```

```
1 i\
\
Linux Distributions Report\

s/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/
y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
```

الآن، نحفظ الملف باسم `distros.sed` ونشغله كالآتي:

```
[me@linuxbox ~]$ sed -f distros.sed distros.txt

Linux Distributions Report

SUSE            10.2     2006-12-07
FEDORA          10       2008-11-25
SUSE            11.0     2008-06-19
UBUNTU          8.04     2008-04-24
FEDORA          8        2007-11-08
SUSE            10.3     2007-10-04
UBUNTU          6.10     2006-10-26
FEDORA          7        2007-05-31
UBUNTU          7.10     2007-10-18
UBUNTU          7.04     2007-04-19
SUSE            10.1     2006-05-11
FEDORA          6        2006-10-24
FEDORA          9        2008-05-13
UBUNTU          6.06     2006-06-01
UBUNTU          8.10     2008-10-30
FEDORA          5        2006-03-20
```

كما لاحظت، قام سكربت sed بعمله على أتم وجه، لكن كيف قام بذلك؟ لنلقِ نظرة أُخرى على السكربت. سنستخدم cat لترقيم الأسطر:

```
[me@linuxbox ~]$ cat -n distros.sed
     1    # sed script to produce Linux distributions report
```

```
2
3    1 i\
4    \
5    Linux Distributions Report\
6
7    s/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/
8    y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
```

السطر الأول من السكربت هو تعليق. كما في العديد من ملفات الإعدادات ولغات البرمجة في نظام لينُكس، فإن التعليق يبدأ برمز "#" ويتبعه أيّة ملاحظة أو شرح. يمكن أن تُدرَج التعليقات في أي مكان في الملف (لكن ليس داخل بعضهم البعض)، وتكمن الفائدة من التعليقات في السماح للأشخاص بفهم أو تعديل السكربت بسهولة.

السطر الثاني هو سطر فارغ. وكما في التعليقات، تستخدم الأسطر الفارغة لزيادة قابلية قراءة النص.

تدعم العديد من تعليمات sed عناوين الأسطر. تُستخدم العناوين لكي تُحدِّد الأسطر التي ستُجرى العمليات عليها. يمكن تحديد عنوان السطر كأرقام أو كمجالات أو باستخدام الرمز "$" لتحديد آخر سطر في الملف؛ كما مرّ معنا في جدولٍ سابق.

تحتوي الأسطر من 3 إلى 6 على النص الذي سيُضاف قبل العنوان 1، أي أول سطر في المدخلات. يتبع التعليمة i محرف العودة إلى بداية السطر (تم تهريبه بالشرطة المائلة الخلفية) أو ما يسمى "محرف إكمال السطر" (line continuation character). تسمح هذه العبارة التي تُستخدم في العديد من الأماكن بما فيها سكربتات الشِل، بأن يضاف محرف العودة إلى بداية السطر في النص دون إخطار المفسر (في هذه الحالة sed) أن السطر قد انتهى (قد يبدو الأمر مربكًا للوهلة الأولى). التعليمة i وشبيهاتها: a (التي تضيف النص بعد السطر الحالي وليس قبله)، و c (التي تستبدل النص)؛ تقبل أن ينتهي جميع الأسطر عدا آخر سطر بمحرف إكمال السطر. السطر السادس في السكربت هو نهاية النص الذي سيُدرَج، الذي ينتهي بمحرف العودة إلى بداية السطر بدلًا من محرف إكمال السطر. مما يشير إلى نهاية التعليمة i.

ملاحظة: يتكون محرف إكمال السطر من شرطة مائلة خلفية يتبعها مباشرةً محرف العودة إلى بداية السطر. لا يُسمَح بأي شيء آخر بعدها وحتى لو كان فراغًا واحدًا.

يحتوي السطر السابع على تعليمة البحث والاستبدال. ولأنها لم تُسبَق بعنوان، فستُنفَّذ على جميع أسطر الملف.

يقوم السطر الثامن بعملية تحويل لجميع الأحرف الصغيرة إلى الأحرف الكبيرة. لاحظ أن التعليمة y في sed لا تسمح باستخدام مجالات الحروف (على سبيل المثال [a-z])، ولا حتى فئات حروف POSIX (على عكس tr). التعليمة y أيضًا غير مسبوقة بعنوان، أي أنها ستُطبَّق على جميع الأسطر.

الأشخاص الذين يحبون sed يحبون أيضًا...

برنامج sed هو برنامج كفؤ، قادر على القيام بعمليات تعديل معقدة جدًا على النصوص. لكنه يستخدم عادة مع تعليمات بسيطة وليس مع سكربتات طويلة. يُفضِّل العديد من المستخدمين استخدام أدوات أُخرى للقيام بعمليات المعالجة الأكثر تعقيدًا. أشهر أداتين هما awk و perl. إنهما تتجاوزان الأدوات البسيطة التي ناقشناها هنا، وتتوسعان إلى حقل لغات البرمجة. تُستخدم perl مكان سكربتات الشِل لإدارة الأنظمة، بالإضافة إلى استخدامها في تطوير الوِب. awk هي لغة مخصصة أكثر. قوتها في قدرتها على معالجة البيانات المجدولة. هي تشابه sed من حيث معالجتها للملفات النصية سطرًا بسطر، وتستخدم طريقة مشابهة لبرنامج sed لتحديد عناوين الأسطر. وعلى الرغم من أن awk و perl خارجتان عن موضوع هذا الكتاب، إلا أنهما مفيدتان جدًا لمستخدم سطر أوامر لينُكس.

aspell

آخر برنامج سنلقي نظرة عليه في هذا الفصل هو aspell، برنامج aspell هو مدقق إملائي تفاعلي؛ وهو نسخة مطورة من برنامج يسمى ispell. وعلى الرغم من أن aspell يُستخدم كثيرًا من البرامج التي تتطلب تدقيقًا إملائيًا؛ إلا أننا نستطيع استخدامه بمفرده عن طريق سطر الأوامر. يستطيع aspell أن يدقق مختلف أنواع النصوص بذكاء، بما فيها مستندات HTML، وبرامج C/C++، ورسائل البريد الإلكتروني وغيرها.

نستخدم aspell بالشكل الآتي لكي ندقق ملفًا نصيًا بسيطًا:

```
aspell check textfile
```

حيث textfile هو اسم الملف الذي سيُدقق. كمثالٍ عملي، سننشئ ملفًا نصيًا باسم foo.txt يحتوي على بعض الأخطاء الإملائية:

```
[me@linuxbox ~]$ cat > foo.txt
The quick brown fox jimped over the laxy dog.
```

سندقق الآن الملف باستخدام aspell:

```
[me@linuxbox ~]$ aspell check foo.txt
```

لما كان aspell مدققًا تفاعليًا، فإننا سنشاهد شاشة كالشاشة الآتية:

```
The quick brown fox jimped over the laxy dog.
```

```
   1) jumped                                          6) wimped
   2) gimped                                          7) camped
   3) comped                                          8) humped
   4) limped                                          9) impede
   5) pimped                                          0) umped
   i) Ignore                                          I) Ignore all
   r) Replace                                         R) Replace all
   a) Add                                             l) Add Lower
   b) Abort                                           x) Exit

   ?
```

سنشاهد في أعلى الشاشة الكلمة التي هُجئت خطأً مُعلّمةً. في المنتصف، سنشاهد عشرة اقتراحات مرقمةً من الصفر إلى التسعة، يليها قائمة بالأفعال التي يمكن القيام بها. وفي الأسفل نشاهد مِحثًا جاهزًا لقبول المدخلات.

إذا ضغطنا على الزر 1، فسيستبدل aspell الكلمة المُعلّمة بالكلمة "jumped" وسيتحرك إلى كلمة أُخرى مهجأة بشكل خاطئ التي هي "laxy". إذا حددنا البديل "lazy" فسيتم إنهاء aspell بعد استبدالها. بعد انتهاء التدقيق الإملائي، نستطيع تفحص محتوى ملف foo.txt وسنجد أن الكلمات الخطأ قد صُحِحَت:

```
[me@linuxbox ~]$ cat foo.txt
The quick brown fox jumped over the lazy dog.
```

إذا لم نحدد الخيار dont-backup--، فسينشئ aspell ملفًا احتياطيًا يحتوي على النص الأصلي وذلك بإضافة الامتداد bak. إلى اسم الملف.

بواسطة مهارتنا في استخدام sed، سنعيد الأخطاء الإملائية السابقة حتى نستطيع تدقيق الملف مرّةً أُخرى:

```
[me@linuxbox ~]$ sed -i 's/lazy/laxy/; s/jumped/jimped/' foo.txt
```

الخيار i- يجعل sed يعدل الملف مباشرةً دون إرساله إلى مجرى الخرج القياسي، وسيستبدل محتوى الملف الأصلي بالناتج المعدل. لاحظنا أيضًا إمكانية وضع أكثر من تعليمة تعديل في نفس السطر؛ وذلك بالفصل بينها بفاصلة منقوطة.

سنجرّب الآن استخدام aspell لتدقيق أنواع مختلفة من الملفات النصية. باستخدام محرر نصي كمحرر vim (ربما يجرّب البعض استخدام sed)، سنكتب مستند HTML:

```
<html>
        <head>
              <title>Mispelled HTML file</title>
        </head>
        <body>
              <p>The quick brown fox jimped over the laxy dog.</p>
        </body>
</html>
```

سنواجه مشكلة عند محاولة تدقيق ملفنا المعدل، إذا استخدمنا الأمر:

```
[me@linuxbox ~]$ aspell check foo.txt
```

فسنحصل على:

```
<html>
        <head>
                   <title>Mispelled HTML file</title>
        </head>
        <body>
               <p>The quick brown fox jimped over the laxy dog.</p>
        </body>
</html>

 1) HTML                                      4) Hamel
 2) ht ml                                     5) Hamil
 3) ht-ml                                     6) hotel


 i) Ignore                                    I) Ignore all
 r) Replace                                   R) Replace all
 a) Add                                       l) Add Lower
 b) Abort                                     x) Exit

 ?
```

سيعتبر aspell أن وسوم HTML هي كلمات مهجأة خطأً. يمكن حلّ هذه المشكلة بتحديد الخيار

‎-H (HTML)، كالآتي:

```
[me@linuxbox ~]$ aspell -H check foo.txt
```

الذي سيظهر النتيجة الآتية:

```
<html>
        <head>
                    <title>Mispelled HTML file</title>
        </head>
        <body>
                <p>The quick brown fox jimped over the laxy dog.</p>
        </body>
</html>

 1)  Mi spelled                              6) Misapplied
 2)  Mi-spelled                              7) Miscalled
 3)  Misspelled                              8) Respelled
 4)  Dispelled                               9) Misspell
 5)  Spelled                                 0) Misled


 i)  Ignore                                  I) Ignore all
 r)  Replace                                 R) Replace all
 a)  Add                                     l) Add Lower
 b)  Abort                                   x) Exit

 ?
```

تم تجاهل جميع وسوم HTML، وستُدقق أي نصوص ليست من الوسوم. لكن ستدقق محتويات الخاصية Alt.

ملاحظة: سيتجاهل aspell روابط الوِب وعناوين البريد الإلكتروني في النصوص. يمكن أن يُعدّل ذلك بخيارات سطر الأوامر. من الممكن أيضًا تحديد أيّة وسوم سيتم تجاهلها. راجع صفحة الدليل للأمر aspell لمزيد من المعلومات.

الخلاصة

ألقينا نظرة في هذا الفصل على العديد من أدوات سطر الأوامر التي تُستخدم لمعالجة النصوص. سنتعرف على غيرها في الفصل القادم. لا يمكننا إنكار أن الفائدة العمليّة من استخدام أدوات معالجة النصوص لم تتضح لك بعد، ولا الطريقة ولا السبب الذي ستستخدمها من أجله؛ على الرغم من أننا حاولنا تجربة أمثلة عملية نستخدم فيها تلك الأدوات. سنكتشف في الفصول اللاحقة أن هذه الأدوات تشكل قاعدةً وأساسًا سنَحُلّ بواسطته الكثير من المشاكل. خصوصًا عند كتابتنا سكربتات الشِل؛ المكان الذي تظهر فيه القيمة الحقيقية لهذه الأدوات.

أضف إلى معلوماتك

هنالك عددٌ من أوامر معالجة النصوص التي تستحق الاهتمام. منها: `split` (تقسيم الملفات إلى أقسام)، و `csplit` (تقسيم الملفات إلى أقسام اعتمادًا على المحتوى).

الفصل الحادي والعشرون:

تنسيق النصوص

سنكمل في هذا الفصل شرحنا للأدوات المتعلقة بالنصوص، مركزين على البرامج التي تُستخدم لتنسيق البيانات النصية عوضًا عن معالجتها. تُستخدم هذه الأدوات عادةً لتحضير النصوص للطباعة، الموضوع الذي سنشرحه في الفصل القادم، البرامج التي سنشرحها في هذا الفصل هي:

- nl - ترقيم الأسطر.
- fold - جعل الأسطر تلتف عند تجاوزها حدًّا مُعينًا.
- fmt - مُنسّق نصوص بسيط.
- pr - تجهيز النص للطباعة.
- printf - تنسيق وإخراج البيانات.
- groff - نظام تنسيق للمستندات.

أدوات التنسيق البسيطة

سنلقي أولًا نظرة على الأدوات البسيطة لتنسيق النصوص. تقوم تلك البرامج عادةً بمهمة واحدة فقط، وتكون آلية عملها سهلة وغير معقدة البتة؛ لكن قد يُستفاد منها استفادةً كبيرةً عند استخدامها في سكربت أو أنبوب.

ترقيم الأسطر باستخدام nl

البرنامج nl هو برنامج بسيط يستخدم للقيام بمهمة واحدة هي ترقيم الأسطر. يشابه nl في أبسط حالاته الأمر cat -n:

```
[me@linuxbox ~]$ nl distros.txt | head
     1	SUSE	10.2	12/07/2006
     2	Fedora	10	11/25/2008
     3	SUSE	11.0	06/19/2008
     4	Ubuntu	8.04	04/24/2008
     5	Fedora	8	11/08/2007
```

```
     6        SUSE       10.3    10/04/2007
     7        Ubuntu     6.10    10/26/2006
     8        Fedora     7       05/31/2007
     9        Ubuntu     7.10    10/18/2007
    10        Ubuntu     7.04    04/19/2007
```

وكما في cat، يقبل nl المدخلات من مجرى الدخل القياسي أو عن طريق ملفات تُمرر أسماؤها كوسائط؛ لكن nl يملك عددًا من الخيارات ويدعم نمط وصفي بسيط للقيام بأنواع معقدة نوعًا ما من الترقيم.

يدعم nl مفهومًا يسمى "الصفحات المنطقية" (logical pages) عند الترقيم. مما يسمح لبرنامج nl بإعادة الترقيم من البداية عند كل صفحة. من الممكن باستخدام الخيارات تحديد قيمة بداية الترقيم، والتحكم في تنسيقه إلى حدٍ ما. تُقسّم الصفحة المنطقية إلى ترويسة، وجسم، وتذييل. يمكن إعادة الترقيم من البداية أو تغيير تنسيق الترقيم في كل قسم من هذه الأقسام. إذا مُرِّرَ إلى nl عدّة ملفات، فستعامل على أنها مجرى وحيد من النص.

يمكن تحديد أقسام النص باستخدام أنماط (غريبة المظهر) تُضاف إلى النص:

الجدول 21-1: أنماط nl

النمط	المعنى
`\:\:\:`	بداية ترويسة الصفحة.
`\:\:`	بداية جسد الصفحة.
`\:`	بداية تذييل الصفحة.

يجب أن يظهر كل نمط من الأنماط السابقة وحده في سطرٍ بأكمله. لن يُظهِر nl هذه الأنماط بعد معالجة الملف.

يحتوي الجدول الآتي على أبرز الخيارات التي تُستخدم مع nl:

الجدول 21-2: خيارات nl الشائعة

الخيار	الشرح
`-b style`	تحديد قيمة نمط ترقيم جسد الصفحة إلى القيمة style؛ حيث style هي إحدى القيم الآتية: • a: ترقيم جميع الأسطر.

	• t: ترقيم جميع الأسطر غير الفارغة، وهو الخيار الافتراضي. • n: عدم ترقيم أي شيء. • pregexp: ترقيم الأسطر التي تُطابِق التعبير النظامي الأساسي regex فقط.
-f style	تحديد نمط ترقيم التذييل إلى style. القيمة الافتراضية هي n.
-h style	تحديد نمط ترقيم الترويسة إلى style. القيمة الافتراضية هي n.
-i number	تحديد قيمة زيادة الترقيم إلى الرقم number. القيمة الافتراضية هي الواحد.
-n format	تحديد تنسيق الرقم إلى format حيث يمكن أن يكون أحد القيم الآتية: • القيمة ln: جعل محاذاة الترقيم إلى اليسار دون طباعة أصفار بادئة. • القيمة rn: جعل محاذاة الترقيم إلى اليمين دون طباعة أصفار بادئة. • القيمة rz: جعل محاذاة الترقيم إلى اليمين مع طباعة أصفار بادئة.
-p	إعادة الترقيم إلى قيمته الابتدائية عند بداية كل صفحة.
-s string	إضافة السلسلة النصية string إلى نهاية كل سطر لإنشاء فاصل. القيمة الافتراضية هي مسافة جدولة واحدة.
-v number	تحديد بداية كل صفحة إلى الرقم number. القيمة الافتراضية هي الواحد.
-w width	تحديد عرض حقل رقم السطر إلى width. القيمة الافتراضية هي ستة.

لا نستطيع أن ننكر أننا لن نستخدم ترقيم الأسطر كثيرًا، لكننا سنستخدم nl لكي نتعلم طريقة دمج عدّة أدوات مع بعضها البعض للقيام بمهام معقدة. سنبني عملنا على مثال من الفصل السابق، لإنشاء تقرير عن توزيعات لينُكس. ولأننا سنستخدم nl، فلا بأس من تحديد الترويسة والجسد والتذييل. سنعدّل سكربت sed من الفصل السابق باستخدام محرر نصي؛ وسنسمي الملف الناتج باسم distros-nl.sed:

```
# sed script to produce Linux distributions report

1 i\
\\:\\:\\:\
\
```

```
Linux Distributions Report\
\
Name            Ver. Released\
----            ---- --------\
\\:\\:
s/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/
$ a\
\\:\
\
End Of Report
```

يضيف السكربت أنماط الصفحة الخاصة بالبرنامج nl ويضيف تذييلًا في آخر التقرير. لاحظ أننا نحتاج إلى استخدام شرطتَين خلفيتَين مائلتَين بدلًا من واحدة، لأن sed يفسرها على أنه محرف الهروب.

سننشئ الآن التقرير باستخدام sort و sed و nl:

```
[me@linuxbox ~]$ sort -k 1,1 -k 2n distros.txt | sed -f distros-nl.sed
| nl


         Linux Distributions Report

         Name       Ver.       Release
         ----       ----       --------

       1  Fedora    5          2006-03-20
       2  Fedora    6          2006-10-24
       3  Fedora    7          2007-05-31
       4  Fedora    8          2007-11-08
       5  Fedora    9          2008-05-13
       6  Fedora    10         2008-11-25
       7  SUSE      10.1       2006-05-11
       8  SUSE      10.2       2006-12-07
       9  SUSE      10.3       2007-10-04
      10  SUSE      11.0       2008-06-19
      11  Ubuntu    6.06       2006-06-01
      12  Ubuntu    6.10       2006-10-26
```

```
      13  Ubuntu    7.04       2007-04-19
      14  Ubuntu    7.10       2007-10-18
      15  Ubuntu    8.04       2008-04-24
      16  Ubuntu    8.10       2008-10-30




         End Of Report
```

التقرير هو عبارة عن نتيجة لأنبوب يحتوي على ثلاثة أوامر. رتبنا أولًا التوزيعات حسب الاسم والإصدار (الحقلين الأول والثاني)، ومن ثم عالجنا الناتج باستخدام sed مضيفين ترويسة التقرير وتذييله. وفي النهاية، مررنا الناتج إلى nl الذي يُرقِّم أسطر جسد الصفحة فقط افتراضيًا.

بإمكاننا إعادة تنفيذ الأمر السابق وتجربة خيارات مختلفة للأمر nl:

```
nl -n rz
```

و:

```
nl -w 3 -s ' '
```

التفاف الأسطر بعد تجاوزها طولًا محددًا باستخدام fold

"الطيّ" (fold)، هو عملية تُقَسِّم السطر عند عرض معين. كما في الأوامر الأخرى، يقبل fold المدخلات من مجرى الدخل القياسي أو من الملفات التي تُمرر أسماؤها كوسائط. يمكننا معرفة كيف يعمل fold بتمرير نص بسيط إليه:

```
[me@linuxbox ~]$ echo "The quick brown fox jumped over the lazy dog." |
fold -w 12
The quick br
own fox jump
ed over the
lazy dog.
```

يُقسّم النص المُمرر من ناتج echo إلى أجزاء باستخدام الخيار ‎-w. حددنا في الأمر السابق، على سبيل المثال، قيمة العرض الأعظمي للسطر بإثني عشر محرفًا. إذا لم يُحدَّد العرض الأعظمي، فستُستخدم القيمة 80. لاحظ كيف قُطِّعَ السطر دون الأخذ بعين الاعتبار حدود الكلمة. بإضافة الخيار ‎-s، نجعل fold يقسم السطر عند آخر

فراغ يصل إليه قبل العرض الأعظمي للسطر:

```
[me@linuxbox ~]$ echo "The quick brown fox jumped over the lazy dog." |
fold -w 12 -s
The quick
brown fox
jumped over
the lazy
dog.
```

برنامج fmt: مُنسّق نصوص بسيط

ينسق برنامج fmt الفقرات ويسمح بالتفاف الأسطر، بالإضافة إلى غيرها من الميزات. يقبل fmt المدخلات من مجرى الدخل القياسي أو من الملفات التي تُمرر أسماؤها كوسائط. بشكل أساسي، يملأ الأسطر بالنص مع المحافظة على المحاذاة والأسطر الفارغة.

سنحتاج إلى بعض النص لكي نجرّب عليه، النص الآتي من صفحة info للأمر fmt:

```
  `fmt' reads from the specified FILE arguments (or standard input
if none are given), and writes to standard output.
   By default, blank lines, spaces between words, and indentation are
preserved in the output; successive input lines with different
indentation are not joined; tabs are expanded on input and introduced
on output.
    `fmt' prefers breaking lines at the end of a sentence, and tries
to avoid line breaks after the first word of a sentence or before the
last word of a sentence. A "sentence break" is defined as either the
end of a paragraph or a word ending in any of `.?!', followed by two
spaces or end of line, ignoring any intervening parentheses or
quotes. Like TeX, `fmt' reads entire "paragraphs" before choosing
line breaks; the algorithm is a variant of that given by Donald E.
Knuth and Michael F. Plass in "Breaking Paragraphs Into Lines",
 `Software--Practice & Experience' 11, 11 (November 1981), 1119-1184.
```

لننسخ هذا النص إلى محررنا النصي ولنحفظه باسم "fmt.info.txt". لنفترض الآن أننا نريد أن نعيد تنسيق هذا النص لكي يتسع في حقل بعرض خمسين محرفًا. يمكننا معالجة الملف باستخدام الأمر fmt مع الخيار "-w":

```
[me@linuxbox ~]$ fmt -w 50 fmt-info.txt | head
    `fmt' reads from the specified FILE arguments
    (or standard input if
none are given), and writes to standard output.
    By default, blank lines, spaces between words,
    and indentation are
preserved in the output; successive input lines
with different indentation are not joined; tabs
are expanded on input and introduced on output.
```

حسنًا، نتيجة غريبة بعض الشيء. ربما علينا قراءة النص، لأنه يشرح ما الذي جرى:

"افتراضيًا، يتم المحافظة على الأسطر الفارغة، والفراغات ما بين الكلمات، والإزاحة في المخرجات؛ لن تُجمَع الأسطر المعاقبة ذات الإزاحة المختلفة؛ ستوسع مسافات الجدولة في المدخلات وستظهر في المخرجات."

إذًا، يحافظ fmt على إزاحة السطر الأول. لحسن الحظ، يوفر fmt خيارًا لتصحيح ذلك:

```
[me@linuxbox ~]$ fmt -cw 50 fmt-info.txt
    `fmt' reads from the specified FILE arguments
(or standard input if none are given), and writes
to standard output.
    By default, blank lines, spaces between words,
 and indentation are preserved in the output;
successive input lines with different indentation
are not joined; tabs are expanded on input and
introduced on output.
   `fmt' prefers breaking lines at the end of a
sentence, and tries to avoid line breaks after
the first word of a sentence or before the
last word of a sentence. A "sentence break"
is defined as either the end of a paragraph
or a word ending in any of `.?!', followed
by two spaces or end of line, ignoring any
intervening parentheses or quotes. Like TeX,
`fmt' reads entire "paragraphs" before choosing
line breaks; the algorithm is a variant of
```

```
that given by Donald E. Knuth and Michael F.
Plass in "Breaking Paragraphs Into Lines",
`Software--Practice & Experience' 11, 11
(November 1981), 1119-1184.
```

أفضل بكثير! حصلنا على النتيجة المطلوبة باستخدام الخيار "-c".

يملك fmt عددًا من الخيارات المثيرة للاهتمام:

الجدول 21-3: خيارات fmt

الخيار	الشرح
`-c`	جعل fmt يعمل في وضع "crown margin". وهذا ما يحفظ الإزاحة لأول سطرين من الفقرة. يتم إزاحة باقي الأسطر لكي تتماشى مع السطر الثاني.
`-p string`	تنسيق الأسطر التي تبدأ بالعبارة "string". بعد التنسيق، ستُضاف محتويات "string" إلى كل سطر تم تنسيقه. يمكن استخدام هذا الخيار لتنسيق النصوص في الأكواد المصدرية. على سبيل المثال، الكثير من لغات البرمجة وملفات الإعدادات تستخدم رمز "#" للإشارة إلى بدء تعليق؛ ويمكن تنسيقها باستخدام `'# ' -p`. راجع المثال في الأسفل.
`-s`	نمط الالتفاف فقط. في هذا النمط، ستلتف الأسطر عند تجاوزها حدًا مُعينًا . لكن الأسطر القصيرة لن تُدمَج مع غيرها. هذا النمط مفيد عند تنسيق نصوص كالأكواد حيث لا يجوز دمج الأسطر مع بعضها.
`-u`	جعل الفراغات موحدة. هذا ما يُطبِّق أسلوب الكُتّاب التقليدي في تنسيق النص. هذا يعني أنه يوجد فراغ واحد بين الكلمات وفراغين بين الجمل. هذا الخيار مفيد عند إزالة "المحاذاة" أي النص الذي أُضيفت الفراغات إليه لكي تتم محاذاته إلى اليمين أو اليسار.
`-w width`	تنسيق النص لكي يتسع في حقل بعرض width من المحارف. القيمة الافتراضية هي 75. لاحظ أن fmt يجعل الأسطر أقصر من ذاك العرض لكي تتم الموازنة ما بين الأسطر.

الخيار p- مثير للاهتمام بشكلٍ كبير. يمكننا باستخدامه أن ننسق أجزاءً محددة من الملف، وذلك بتزويد هذا الخيار بالعبارة التي تبدأ فيها تلك الأسطر. تستخدم العديد من لغات البرمجة رمز "#" للإشارة إلى بداية

التعليقات، وبالتالي يمكن تنسيق التعليقات باستخدام هذا الخيار. لنُنشئ ملفًا يُحاكي برنامجًا يحتوي على تعليقات:

```
[me@linuxbox ~]$ cat > fmt-code.txt
# This file contains code with comments.

# This line is a comment.
# Followed by another comment line.
# And another.

This, on the other hand, is a line of code.
And another line of code.
And another.
```

الملف السابق يحتوي على تعليقات تبدأ بالسلسلة النصية " #" (أي رمز # تبعه فراغ) وأسطر أُخرى من "الأكواد". سنستخدم الآن fmt لتنسيق التعليقات وترك باقي الأكواد دون أن تُنسّق:

```
[me@linuxbox ~]$ fmt -w 50 -p '# ' fmt-code.txt
# This file contains code with comments.

# This line is a comment. Followed by another
# comment line. And another.

This, on the other hand, is a line of code.
And another line of code.
And another.
```

لاحظ أن التعليقات المتتابعة قد أُضيفت إلى بعضها البعض، بينما تُرِكت الأسطر الفارغة والأسطر التي لا تبدأ بالسلسلة النصية " #" على حالها.

تنسيق النص للطباعة باستخدام pr

يستخدم برنامج pr لكي يقوم "بترقيم الصفحات" (paginate). عند طباعة نص، من المهم فصل الصفحات عن بعضها بعدّة أسطر فارغة، لكي يُنشأ حاشية عليا وسفلى لكل صفحة. وحتى أكثر من ذلك، حيث تستخدم الأسطر الفارغة لتحديد ترويسة وتذييل لكل صفحة.

سنشرح البرنامج pr بتحويل ملف distros.txt إلى سلسلة من الصفحات القصيرة جدًا (عرضنا هنا أول

صفحتين فقط):

```
[me@linuxbox ~]$ pr -l 15 -w 65 distros.txt



2008-12-11 18:27              distros.txt           Page 1




SUSE          10.2    12/07/2006
Fedora        10      11/25/2008
SUSE          11.0    06/19/2008
Ubuntu        8.04    04/24/2008
Fedora        8       11/08/2007



2008-12-11 18:27            distros.txt             Page 2




SUSE          10.3    10/04/2007
Ubuntu        6.10    10/26/2006
Fedora        7       05/31/2007
Ubuntu        7.10    10/18/2007
Ubuntu        7.04    04/19/2007
```

استخدمنا في المثال السابق الخيار l- (طول الصفحة)، والخيار w- (عرض الصفحة)، لتعريف أن "الصفحة" هي خمسٌ وستون محرفًا في العرض وخمسة عشر سطرًا في الطول. عندما يرقم برنامج pr صفحات ملف distros.txt، فإنه يفصل بين كل صفحة وأُخرى بعدّة فراغات بيضاء ويُنشئ ترويسةً افتراضيةً تحتوي على تاريخ تعديل الملف، واسم الملف، ورقم الصفحة. يوفر برنامج pr العديد من الخيارات للتحكم في طريقة تخطيط الصفحة. سنلقي نظرة عليها في الفصل القادم.

printf: تنسيق وإخراج البيانات

على النقيض من باقي الأوامر التي ناقشناها في هذا الفصل، لا يُستخدم الأمر `printf` في الأنابيب (لأنه لا يقبل دخل قياسي) وليس له استخدام مباشر في سطر الأوامر (يُستخدم غالبًا في السكربتات). إذا لماذا هو مهم؟ لأنه شائع الاستخدام.

طُوِّر الأمر `printf` (جاء اسمه من العبارة "print formatted") لأول مرّة للغة البرمجة C، وأُستخدم فيما بعد في العديد من لغات البرمجة بما فيها الشِل. في الواقع، إن `printf` هو أمر مُضمَّن في الصدفة bash.

يعمل `printf` كما يلي:

```
printf "format" arguments
```

يُستخدم الأمر بتحديد سلسلة نصية تحتوي على نمط التنسيق الذي سيُطبَّق على الوسائط. ستُخرَج النتيجة المُنسَّقة إلى مجرى الخرج القياسي. هذا مثال مبسط عن استخدامه:

```
[me@linuxbox ~]$ printf "I formatted the string: %s\n" foo
I formatted the string: foo
```

قد تحتوي جملة التنسيق على نصوص عادية (كالعبارة "I formatted the string:")، وعبارة مُهرَّبة (كما في محرف نهاية السطر \n)، وعبارات تبدأ بالرمز %، التي تسمى "محددات التحويل" (conversion specifications) ويقال عنها أيضًا "محددات التنسيق". في المثال أعلاه، يُستخدم محدد التنسيق %s لتنسيق السلسلة النصية "foo" ويضعها في موضعها المناسب في مخرجات الأمر `printf`. هذا مثال آخر:

```
[me@linuxbox ~]$ printf "I formatted '%s' as a string.\n" foo
I formatted 'foo' as a string.
```

كما تلاحظ، أُستبدل محدد التنسيق %s بالسلسلة النصية "foo" في مخرجات الأمر. يُستخدم المحدد "s" لتنسيق البيانات النصية. يوجد هنالك محددات أخرى لتنسيق أنواع البيانات الأخرى. يلخص الجدول الآتي أبرز أنواع البيانات المستخدمة بكثرة:

الجدول 21-4: محددات التنسيق الشائعة

المحدد	الشرح
d	تنسيق عدد ما كعدد صحيح.
f	تنسيق عدد ما كعدد ذي فاصلة عشرية.
o	تحويل وإخراج عدد ما إلى النظام الثماني.

s	تنسيق سلسلة نصية.
x	تحويل وإخراج عدد ما إلى النظام الست عشري باستخدام احرف صغيرة "a-f" عند الحاجة إليها.
X	يقوم بنفس عمل المحدد x لكن باستخدام الأحرف الكبيرة "A-F".
%	طباعة الرمز % (أي أن يُكتَب %%).

سنشرح استخدامات المحددات السابقة على السلسلة النصية "380":

```
[me@linuxbox ~]$ printf "%d, %f, %o, %s, %x, %X\n" 380 380 380 380 380
380
380, 380.000000, 574, 380, 17c, 17C
```

ولأننا استخدمنا ستة محددات تنسيق في عبارة التنسيق، فسنحتاج إلى توفير ستة وسائط. تُظهِر النتائج الستة السابقة تأثير كل من المحددات.

يوجد هنالك عدّة مكونات اختيارية يمكن إضافتها إلى محددات التنسيق لتعديل سلوكها. الآتي هو شكل مُحَدِّد التنسيق الكامل مع جميع مكوناته الاختيارية:

%[*flags*][*width*][.*precision*] conversion_specification

يجب أن تُرتَّب المكونات ترتيبًا صحيحًا عند استخدام أكثر من مكون في نفس المحدد لتفسيرها تفسيرًا صحيحًا. هذا شرح عن كلٍ من المكونات السابقة:

الجدول 21-5: مكونات محدد التنسيق

المُكوّن	الشرح
flags	يمكن استخدام الرايات الخمس الآتية: • #: استخدام "الصيغة البديلة" للمخرجات. يختلف تأثير هذا الخيار باختلاف أنواع البيانات. لمحدد o (إظهار الأرقام في النظام الثماني)، سيتم إسباق الرقم المُخرَج بصفر. لمحددَي X و x (إظهار الأرقام في النظام الست عشري)، سيتم إسباق الرقم المُخرَج بالعبارتين 0X أو 0x على التوالي. • 0 (الرقم صفر): استخدام الرقم 0 كحاشية عند الإخراج. هذا يعني أن الحقل سيُملأ بأصفار كما في "000380".

	• – (الشرطة): جعل محاذاة المخرجات على اليسار. يقوم `printf` افتراضيًا بمحاذاة المخرجات على اليمين. • " " (فراغ واحد): إسباق الأرقام الموجبة بفراغ. • + (إشارة الزائد): إظهار الإشارة للأرقام الموجبة. يُظهِر `printf` الإشارة للأرقام السالبة فقط افتراضيًا.
`width`	رقم يحدد عرض الحقل الأعظمي.
`.precision`	تحديد عدد الأرقام التي ستظهر بعد الفاصلة للأرقام العشرية. أما للسلاسل النصية، فيحدد هذا المكون عدد المحارف التي ستُطبع.

هذه بعض الأمثلة عن استخدام التنسيقات:

الجدول 21-6: أمثلة عن مكونات محددات التنسيق

الوسيط	التنسيق	النتيجة	الشرح
380	"%d"	380	تنسيق بسيط للأعداد الصحيحة.
380	"%#x"	0x17c	عدد صحيح مُنسَّق بنظام العد الست عشري مع استخدام الصيغة البديلة.
380	"%05d"	00380	عدد صحيح مُنسَّق في حقل بسعة خمسة محارف مسبوقًا برقم "0" كحاشية، وتحديد 5 كأصغر عرض للحقل.
380	"%05.5f"	380.00000	عدد عشري مُنسَّق مع إظهار خمسة أرقام بعد الفاصلة وإظهار الحاشية، ولأن سعة الحقل العظمى هي 5 وهي أقل من عدد أرقام الناتج، فلن يكون للحاشية أي تأثير على المخرجات.
380	"%010.5f"	0380.00000	سيتم إظهار الحاشية بعد زيادة سعة الحقل العظمى إلى 10.
380	"%+d"	+380	إظهار إشارة "+" بجانب الرقم الموجب.

380	"%-d"	380	تقوم الراية "-" بمحاذاة النص إلى اليسار.
abcedfghijk	"%5s"	abcedfghijk	تحديد سعة الحقل الدنيا إلى 5.
abcedfghijk	"%.5s"	abcde	تحديد سعة الحقل العظمى إلى 5.

وكما ذكرنا سابقًا، يُستخدم printf كثيرًا في السكربتات لتنسيق البيانات المجدولة وليس عبر سطر الأوامر مباشرةً. لكن ما زلنا نستطيع استخدمه لحلّ بعض مشاكل التنسيق. لنخرج أولًا بعض الحقول مفصولةً بمسافة جدولة tab:

```
[me@linuxbox ~]$ printf "%s\t%s\t%s\n" str1 str2 str3
str1 str2 str3
```

بإدراج t\ (عبارة الهروب التي تُمثل مسافة الجدولة)، استطعنا إنشاء التنسيق المطلوب. سنطبع الآن الأرقام بتنسيق أنيق:

```
[me@linuxbox ~]$ printf "Line: %05d %15.3f Result: %+15d\n" 1071
3.14156295 32589
Line: 01071           3.142 Result:           +32589
```

يُظهر المثال السابق تأثير القيمة الدنيا لعرض الحقل في المسافات ما بين الحقول. ماذا عن تنسيق صفحة وِب صغيرة:

```
[me@linuxbox ~]$ printf "<html>\n\t<head>\n\t\t<title>%s</title>\n
\t</head>\n\t<body>\n\t\t<p>%s</p>\n\t</body>\n</html>\n" "Page Title"
"Page Content"
<html>
        <head>
                <title>Page Title</title>
        </head>
        <body>
                <p>Page Content</p>
        </body>
</html>
```

نظم تنسيق المستندات

تفحصنا إلى الآن العديد من الأدوات البسيطة لتنسيق النصوص. هذه الأدوات مفيدة للمهام الصغيرة والبسيطة، لكن ماذا عن الأعمال الكبيرة؟ أحد الأسباب التي جعلت يونكس نظام تشغيل شهير بين المستخدمين التقنيين والباحثين العلميين (بجانب توفير بيئة متعددة المستخدمين والمهام، ممتازة لتطوير البرمجيات) هو توفر الأدوات التي تسمح ببناء مختلف أنواع المستندات وخصوصًا المنشورات العلمية والأكاديمية. في الواقع، يعتبر توثيق غنو أن آلية تنسيق المستندات هي السبب الحقيقي لتطوير يونكس:

> "طُوِّرت أول نسخة من يونكس على جهاز PDP-7 الذي كان موجودًا في مختبرات `bell`. في 1971، أراد المطورون الحصول على جهاز PDP-11 لإكمال العمل على نظام التشغيل. ولكي يستحق النظام ثمنه، اقترح المطورون تضمين نظام تنسيق للمستندات لقسم براءات الاختراع في شركة AT&T. أول برنامج للتنسيق كان مبنيًا على برنامج '`roff`'، وكُتِبَ بواسطة `J. F. Ossanna`."

توجد هنالك عائلتان أساسيتان من منسقات الوثائق تُسيطران على هذا المجال: الأدوات التي تنحدر من برنامج `roff` الأصلي بما فيها `nroff` و `troff`، والأدوات المبنية على نظام التنضيد $\TeX$ (تُلفظ "تِك"). نعم، حرف "E" النازل تحت السطر هو جزء من اسم البرنامج!

جاء الاسم "`roff`" من المصطلح "run off" كما في الجملة الإنكليزية "I'll run off a copy for you". يُستخدم برنامج nroff لتنسيق المستندات لإخراجها إلى الأجهزة التي تستخدم الخطوط ذات العرض الثابت (monospaced fonts)، كالطرفيات والطابعات التي تتبع أسلوب الآلات الكاتبة. كانت هذه الأداة تدعم جميع الأجهزة التي قد توصل بالحاسوب التي كانت موجودة عندما أُنشئت تلك الأداة. أُنشئ البرنامج `troff` لدعم تنسيق الملفات التي تستخدمها الطابعات التجارية الحديثة (آنذاك). تحتوي عائلة أدوات `roff` على عدّة برامج أخرى تُستخدم لتنسيق أقسام من المستندات. تتضمن هذه البرامج eqn (للمعادلات الرياضية) و `tbl` (للجداول).

ظهر برنامج $\TeX$ لأول مرّة (بإصدارٍ مستقر) في عام 1989، وأدى، إلى حدٍ ما، إلى إزاحة `troff` من مكانته. لكن لن نشرح استخدام $\TeX$ هنا لسبَبين: الأول، تعقيده (توجد كتب كاملة تتحدث عنه)، والثاني، لعدم وجوده افتراضيًا على أغلب توزيعات لينُكس الحديثة.

تلميحة: للمهتمين بتثبيت $\TeX$، جربو الحزمة `texlive` الموجودة في مستودعات أغلب التوزيعات، والبرنامج الرسومي LyX.

---

groff

إن `groff` هو حزمة من البرامج تحتوي على نسخة غنو من `troff`. وتحتوي على سكربت يجعله يحاكي

nroff وباقي عائلة roff أيضًا.

بينما يُستخدم برنامج roff والبرامج المبنية عليه لإنشاء مستندات مُنسَّقة، لكنهم يقومون بذلك بطريقة غريبة جدًا بالنسبة إلى المستخدمين العصريين. تُصنَع أغلب المستندات اليوم باستخدام برامج معالجة النصوص التي نستطيع فيها الكتابة مع التنسيق بخطوة واحدة، قبل وجود برامج معالجة النصوص، كانت عملية إنتاج المستندات تتألف من خطوتَين: كتابة الملف باستخدام محرر نصي، وتنسيق الملف بأحد البرامج كبرنامج troff. اللغة الوصفية التي تحتوي على التعليمات التي يستخدمها برنامج التنسيق كانت تُضمَّن داخل الملف النصي. مثال عن تلك العملية هو مستندات الوِب، التي تُبنى باستخدام محرر نصي ومن ثم تُعالَج باستخدام متصفح وِب وتظهر النتيجة النهائية.

لن نشرح برنامج groff كاملًا، لأن العديد من عناصر لغته الوصفية تُستخدم لبعض التفاصيل الصغيرة التي ليس لها قيمة. لكننا سنركز على واحدة من حزم الماكرو (macro packages) التي ما تزال تستخدم كثيرًا إلى يومنا هذا. هذه الحزم تلخص الأوامر ذات المستوى المنخفض إلى مجموعة أصغر من الأوامر عالية المستوى، مما يُسَهِّل استخدام groff لدرجة كبيرة.

لنأخذ بعين الاعتبار صفحة دليل بسيطة، موجودة في مجلد usr/share/man/ كملف نصي مضغوط ببرنامج gzip. إذا حاولنا مشاهدة محتواها (بعد فك ضغطها طبعًا)، سنشاهد الآتي (صفحة الدليل للأمر ls في القسم الأول من صفحات الدليل):

```
[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | head
  .\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.35.
  .TH LS "1" "April 2008" "GNU coreutils 6.10" "User Commands"
  .SH NAME
  ls \- list directory contents
  .SH SYNOPSIS
  .B ls
  [\fIOPTION\fR]... [\fIFILE\fR]...
  .SH DESCRIPTION
  .\" Add any additional description here
  .PP
```

بمقارنة الناتج السابق بصفحة man الحقيقية، فسوف نبدأ بمعرفة العلاقة ما بين اللغة الوصفية ومخرجاتها:

```
[me@linuxbox ~]$ man ls | head
LS(1)                           User Commands                      LS(1)
```

```
NAME
        ls - list directory contents
SYNOPSIS
           ls [OPTION]... [FILE]...
```

السبب وراء ظهور صفحة الدليل بهذا الشكل هو معالجتها باستخدام groff، عن طريق حزمة الماكرو mandoc. يمكننا أن نحاكي الأمر man بالأنبوب الآتي:

```
[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | groff -mandoc -T
ascii | head
LS(1)                          User Commands                         LS(1)



NAME
        ls - list directory contents
SYNOPSIS
           ls [OPTION]... [FILE]...
```

استخدمنا في المثال السابق الأمر groff مع بعض الخيارات لتحديد حزمة الماكرو mandoc وتحديد نمط ASCII كمخرجات. يستطيع groff أن يقوم بالإخراج إلى عدّة صيغ أو أنماط. سيستخدم PostScript افتراضيًّا إذا لم تُحَدَّد الصيغة:

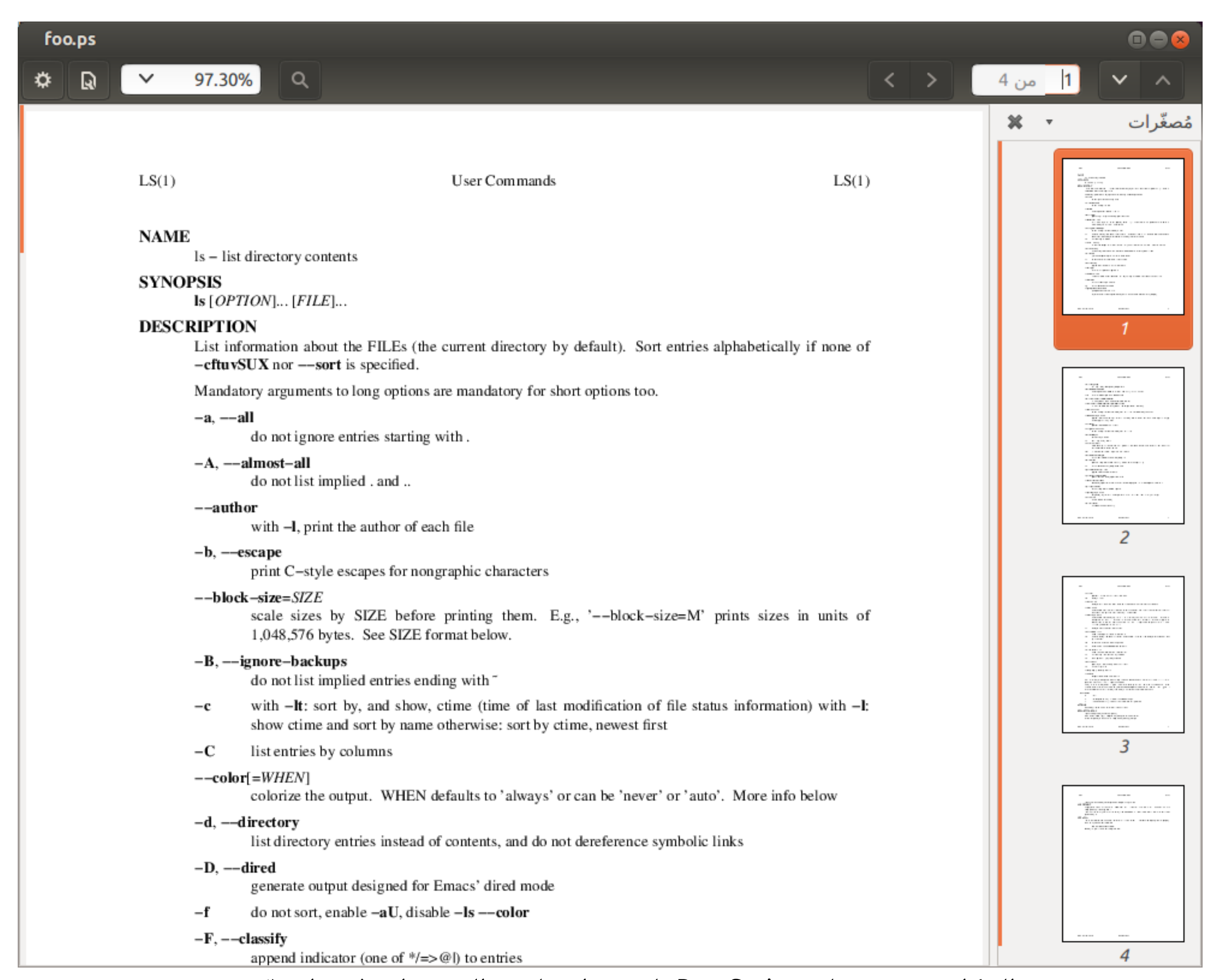
```
[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | groff -mandoc |
head
%!PS-Adobe-3.0
%%Creator: groff version 1.18.1
%%CreationDate: Thu Feb 5 13:44:37 2009
%%DocumentNeededResources: font Times-Roman
%%+ font Times-Bold
%%+ font Times-Italic
%%DocumentSuppliedResources: procset grops 1.18 1
%%Pages: 4
%%PageOrder: Ascend
%%Orientation: Portrait
```

لقد ذكرنا PostScript باختصار في الفصل السابق، وكذلك سنفعل في الفصل القادم. PostScript هو لغة لوصف

الصفحات تستخدم لوصف محتويات صفحة مطبوعة إلى الطابعات. إذا حفظنا محتويات الأمر السابق إلى ملف (على فرض أننا نستخدم سطح مكتب رسومي ولدينا المجلد Desktop):

```
[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | groff -mandoc >
~/Desktop/foo.ps
```

ستظهر أيقونة على سطح المكتب تحتوي على المخرجات. سيُفتَح عارض المستندات بالنقر المزدوج على الأيقونة، وستُعرَض محتويات الملف:



الشكل 4: عرض ملف PostScript باستخدام عارض المستندات في واجهة غنوم

من الممكن أيضًا تحويل ملف PostScript إلى PDF (صيغة الملفات المحمولة أو protable document format) باستخدام هذا الأمر:

```
[me@linuxbox ~]$ ps2pdf ~/Desktop/foo.ps ~/Desktop/ls.pdf
```

البرنامج ps2pdf هو جزء من حزمة ghostscript، المُثبَّتة على أغلب توزيعات لينُكس التي تدعم الطباعة.

تلميحة: تحتوي توزيعات لينُكس عادةً على العديد من برامج سطر الأوامر التي تستخدم لتحويل صيغ الملفات. عادةً ما يكون اسمها على الشكل format2format. جرّب الأمر:

```
ls /usr/bin/*[[:alpha:]]2[[:alpha:]]*
```

لكي تتعرف عليها. جرّب أيضًا البحث عن البرامج المسماة format**to**format.

سنزور صديقنا القديم distros.txt كآخر تمرين لنا مع groff. هذه المرّة سنستخدم البرنامج tbl لتنسيق جدول التوزيعات. سنعدل سكربت sed لإضافة اللغة الوصفية، ومن ثم سنمرر الناتج إلى groff.

يجب علينا أولًا أن نقوم بالتعديلات الضرورية على سكربت sed التي يتطلبها tbl. سنغيّر محتوى ملف distros.sed إلى الآتي، باستخدام محرر نصي:

```
# sed script to produce Linux distributions report

1 i\
.TS\
center box;\
cb s s\
cb cb cb\
l n c.\
Linux Distributions Report\
=\
Name Version Released\
_
s/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/
$ a\
.TE
```

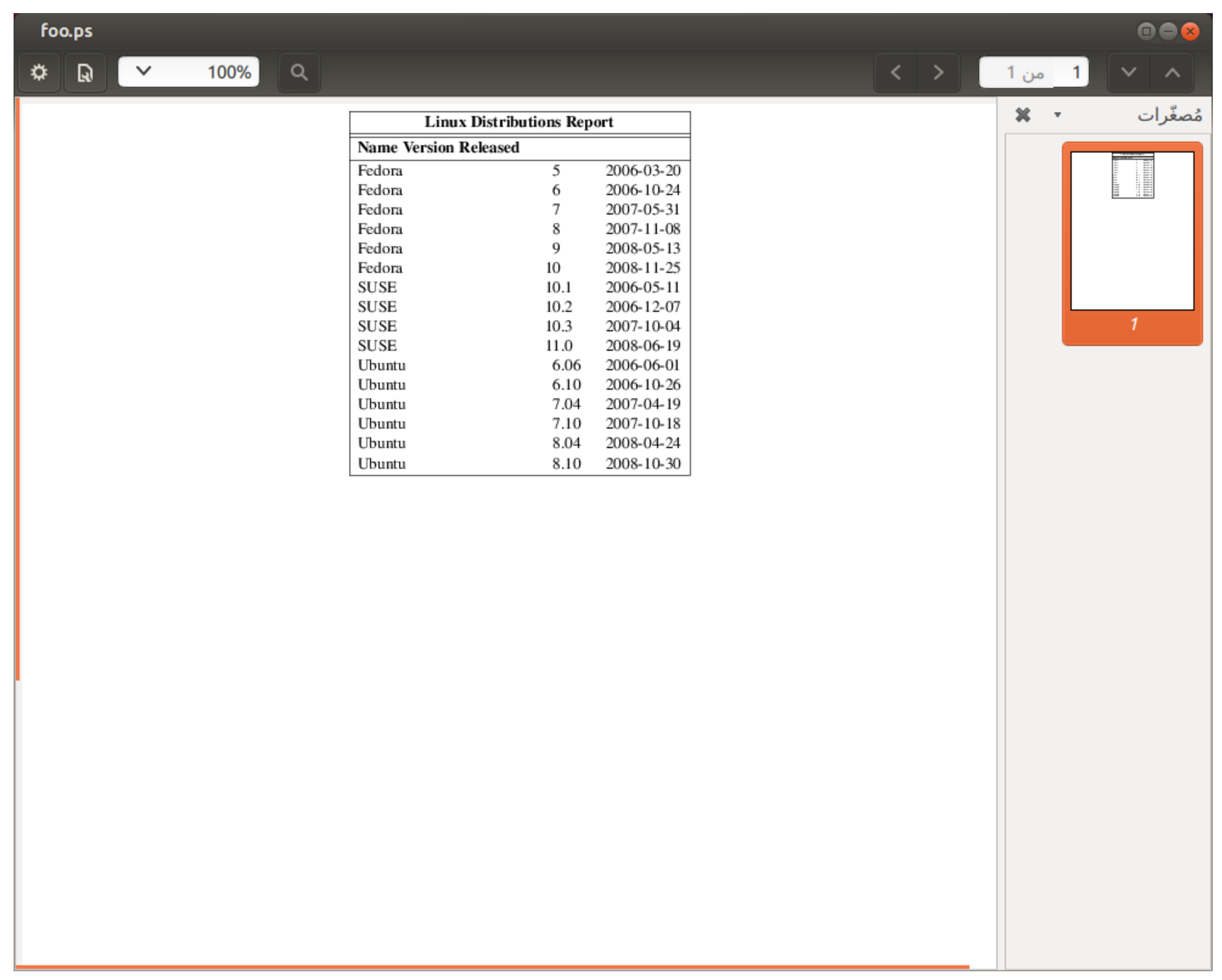
لكي يعمل السكربت السابق دون مشاكل، لاحظ أن الكلمات "Name Version Released" مفصولة بمسافات جدولة وليس بفراغات. سنحفظ الملف الناتج بالاسم distros-tbl.sed. يستخدم tbl الوسمَين ".TS" و ".TE" للإشارة إلى بداية ونهاية الجدول. الأسطر التي تحتوي على وسم TS. تُعَرِّف الخصائص العامة للجدول، التي هي -في مثالنا السابق- توسيط الجدول أفقيًا في الصفحة، وإنشاء إطار له. الأسطر البقية تحدد تخطيط الجدول. لو جربنا الآن سكربت sed الجديد، فسوف نحصل على النتيجة الآتية:

```
[me@linuxbox ~]$ sort -k 1,1 -k 2n distros.txt | sed -f distros-tbl.sed
| groff -t -T ascii 2>/dev/null
                       +------------------------------------+
                       | Linux Distributions Report         |
                       +------------------------------------+
                       | Name      Version       Released   |
                       +------------------------------------+
                       |Fedora         5        2006-03-20 |
                       |Fedora         6        2006-10-24 |
                       |Fedora         7        2007-05-31 |
                       |Fedora         8        2007-11-08 |
                       |Fedora         9        2008-05-13 |
                       |Fedora         10       2008-11-25 |
                       |SUSE          10.1      2006-05-11 |
                       |SUSE          10.2      2006-12-07 |
                       |SUSE          10.3      2007-10-04 |
                       |SUSE          11.0      2008-06-19 |
                       |Ubuntu         6.06     2006-06-01 |
                       |Ubuntu         6.10     2006-10-26 |
                       |Ubuntu         7.04     2007-04-19 |
                       |Ubuntu         7.10     2007-10-18 |
                       |Ubuntu         8.04     2008-04-24 |
                       |Ubuntu         8.10     2008-10-30 |
                       +------------------------------------+
```

إضافة الخيار t- إلى برنامج groff تجعله يعالج النص باستخدام tbl. وبشكل مشابه لأحد الأمثلة السابقة، يُستخدم الخيار T- لإظهار المخرجات بنمط ASCII بدلًا من النمط الافتراضي PostScript.

ستكون المخرجات أفضل بكثير إن لم نجعلها مقتصرةً على شاشة الطرفية فقط. سنحصل على نتيجة مرضية إذا حددنا PostScript كنمط للمخرجات وشاهدنا الملف الناتج بعارض مستندات رسومي:

```
[me@linuxbox ~]$ sort -k 1,1 -k 2n distros.txt | sed -f distros-tbl.sed
| groff -t > ~/Desktop/foo.ps
```



Linux Distributions Report

Name	Version	Released
Fedora	5	2006-03-20
Fedora	6	2006-10-24
Fedora	7	2007-05-31
Fedora	8	2007-11-08
Fedora	9	2008-05-13
Fedora	10	2008-11-25
SUSE	10.1	2006-05-11
SUSE	10.2	2006-12-07
SUSE	10.3	2007-10-04
SUSE	11.0	2008-06-19
Ubuntu	6.06	2006-06-01
Ubuntu	6.10	2006-10-26
Ubuntu	7.04	2007-04-19
Ubuntu	7.10	2007-10-18
Ubuntu	8.04	2008-04-24
Ubuntu	8.10	2008-10-30

الشكل 5: عرض الجدول النهائي

الخلاصة

يرجع سبب وجود أدوات عديدة لمعالجة وتنسيق النصوص في الأنظمة الشبيهة بيونكس إلى كثرة استخدام النصوص فيها. أبسط أدوات تنسيق النصوص كبرنامجي `fmt` و `pr` يُستخدمان في السكربتات لإنتاج مستندات قصيرة، يمكن استخدام `groff` (وأصدقاءه) لكتابة كتب كاملة. ربما لن تكتب مستند تقني باستخدام أدوات سطر الأوامر (على الرغم من العديد من الأشخاص يقومون بذلك!)، لكن من الجيد معرفة إمكانية ذلك.

الفصل الثاني والعشرون:

الطباعة

بعد أن قضينا آخر فصلين نتحدث فيهما عن معالجة النصوص، حان الوقت الآن لكي نضع هذا النص على الورق. سنلقي في هذا الفصل نظرةً على أدوات سطر الأوامر التي تُستخدم لطباعة الملفات والتحكم في عملية الطباعة. لن نشرح طريقة تهيئة الطابعة في هذا الفصل، لأنها تختلف من توزيعة لأُخرى وتُضبَط غالبًا تلقائيًا أثناء التثبيت. لاحظ أننا سنحتاج إلى طابعة مُعدَّة مُسبقًا على جهازك لتنفيذ تمارين هذا الفصل.

سنناقش الأوامر الآتية:

- `pr` – تحويل النصوص للطباعة.
- `lpr` – طباعة الملفات.
- `a2ps` – تنسيق الملفات لطباعتها على طابعة تدعم PostScript.
- `lpstat` – عرض معلومات الحالة للطابعة.
- `lpq` – عرض حالة الطلبية.
- `lprm` – إلغاء أعمال الطباعة.

موجز عن تاريخ الطباعة

لكي نستطيع فهم ميزات الطباعة الموجودة في الأنظمة الشبيهة بِيونكس فهمًا كاملًا؛ يجب علينا أن نتعرف على بعض التاريخ أولًا. يعود تاريخ الطباعة في الأنظمة الشبيهة بِيونكس إلى بداية وجود هذه الأنظمة. لكن كانت طريقة استخدام الطابعات في ذاك الوقت تختلف كثيرًا عما هي عليه حاليًا.

الطباعة في العصور القديمة

كانت الطابعات في عصور ما قبل الحاسوب الشخصي، كما كانت الحواسيب نفسها، كبيرة، وباهظة الثمن، ومركزية. كان يعمل المستخدم العادي للحاسوب في ثمانينات القرن الماضي على طرفية موصولة إلى جهاز بعيد عنها. كانت الطابعة موجودة إلى جوار الحاسوب وتحت رقابة المشرفين عليه.

عندما كانت الطابعات غالية الثمن ومركزية، كما كانوا في الأيام الأولى لنظام يونكس؛ كان من الشائع أن يتشارك عدّة مستخدمين بطابعة واحدة. لكي يتم التعرف على مهام الطابعة التي تنتمي إلى مستخدم

معين، فكانت تطبع ورقة تحتوي على اسم المستخدم في بداية كل مهمة طباعة. ثم يقوم موظفو الدعم الفني بتعبئة عربة تحتوي على مهام الطباعة لليوم بأكمله ويسلمون كل مستخدم الأوراق التي طبعها.

طابعات الحروف

تقنيات الطابعات في الثمانينات تختلف عن التقنيات الحالية بسمتَين أساسيتَين: الأولى، أن الطابعات في تلك الفترة كانت طابعات ميكانيكية تستخدم جزءًا معدنيًا يحتوي على حبر لطباعة الحروف على كل صفحة. أشهر تقنيتان في ذلك الزمن هما: الطباعة عن طريق عجلة الحروف، والطباعة عن طريق مصفوفة النقط.

السمة الثانية والأهم من سمات الطابعات القديمة هي أن الطابعات تستخدم مجموعة ثابتة من المحارف تكون مدمجة مع الطابعة نفسها. على سبيل المثال، طابعة تستخدم عجلة الحروف تستطيع طباعة الحروف الموجودة داخل العجلة فقط، ولا تستطيع طباعة أي شيء آخر؛ وهذا ما يجعل تلك الأنواع من الطابعات تعمل وكأنها آلات كاتبة سريعة جدًا. وكما أغلب الآلات الكاتبة، فكانت تلك الأنواع من الطابعات تطبع الحروف باستخدام خطوط ذات عرض ثابت (monospaced fonts). هذا يعني أن لكل حرف نفس عرض باقي الحروف. وكانت الطباعة تتم في أماكن ثابتة من الصفحة، وكانت المساحة القابلة للطباعة في الصفحة تحتوي على عدد ثابت من الحروف. أغلب الطابعات تطبع عشرة حروف في البوصة (CPI) أفقيًا، وستة أسطر في البوصة (LIP) عموديًا. وهذا ما جعل الورقة من قياس "US Letter" تحتوي على 66 سطرًا و 85 حرفًا في كل سطر. وبعد الأخذ بعين الاعتبار هامش صغير على جانبَيّ الورقة، فكان يعتبر الحد الأقصى للحروف في السطر الواحد هو 80 حرفًا؛ وهذا ما يُفسِّر لماذا تكون شاشات الطرفيات (وعرض نافذة محاكيات الطرفيات من بعدها) تتسع افتراضيًا لثمانين حرفًا. لأنها كانت تحاكي طريقة عرض WYSIWYG (What You See Is What You Get أو "ما تراه هو ما تحصل عليه") للمخرجات عند طباعتها باستخدام خط ذي عرضٍ ثابت.

تُرسَل البيانات إلى طابعة شبيهة بالآلة الكاتبة على شكل سلسلة من البايتات تحتوي على المحارف التي ستُطبَع. على سبيل المثال، لكي يُطبَع الحرف "a"، فإن كود ASCII الذي سيُرسَل هو 97. بالإضافة إلى ذلك، يمكن استخدام أكواد التحكم الموجودة في ASCII لتوفير طرق لتحريك حاملة الحروف (carriage) أو الورقة، وذلك باستخدام محرف العودة إلى بداية السطر (يقصد به عودة حاملة الحروف إلى بداية السطر، وهذا ما يُفسِّر معناه ولماذا يُستخدم)، السطر الجديد ...إلخ. يمكن القيام ببعض تأثيرات النصوص البسيطة باستخدام أكواد التحكم، كالخط العريض على سبيل المثال؛ وذلك بطباعة الحرف ومن ثم فراغ خلفي (backspace تبين لنا هنا أيضًا سبب التسمية) ومن ثم إعادة طباعة الحرف مرّة أخرى للحصول على طباعة بلون أغمق. يمكننا ملاحظة ذلك بفتح صفحة من صفحات الدليل man باستخدام `nroff` ومن ثم معاينة الناتج باستخدام `cat -A`:

```
[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | nroff -man | cat -A
| head
```

```
LS(1)                        User Commands                        LS(1)
$
$
$
N^HNA^HAM^HME^HE$
         ls - list directory contents$
$
S^HSY^HYN^HNO^HOP^HPS^HSI^HIS^HS$
         l^Hls^Hs [_^HO_^HP_^HT_^HI_^HO_^HN]... [_^HF_^HI_^HL_^HE]...$
```

يُستخدم محرف ‎^H (Control-h)‎، الذي يُمثِّل الفراغات الخلفية، لإنشاء تأثير الخط العريض. وبشكل مشابه، نستطيع استخدام الفراغ الخلفي والشرطة السفلية لإنشاء تأثير النص الذي تحته خط.

الطابعات الرسومية

أدى تطوير واجهة المستخدم الرسومية إلى حدوث تغيرات كبيرة في تقنيات الطباعة. وكما انتقلت الحواسيب إلى شاشات تظهر الصور الرسومية (على النقيض من الطرفيات التي لا تظهر سوى النصوص)، فتحولت الطباعة من الطباعة باستخدام الحروف فقط إلى طباعة رسومية. وساعد على ذلك تطوير طابعات الليزر منخفضة الثمن، التي كانت تطبع نقط صغيرة على أي مكان في المنطقة القابلة للطباعة في الورقة بدلًا من طباعة حروف ثابتة فقط. هذا الأمر جعل من الممكن طباعة النصوص بأي خط (وليس فقط الخطوط ذات العرض الثابت)، وحتى طباعة الصور والرسومات عالية الدقة.

لكن الانتقال من الطابعات التي تعتمد على الحروف إلى الطابعات الرسومية شكّل تحديًا تقنيًا صعبًا. هذا هو السبب: يُحسب عدد البايتات اللازم لملء الصفحة بالحروف بالطريقة الآتية (باعتبار أن الصفحة تتسع لستين سطرًا وكل سطر يحتوي على ثمانين محرفًا):

```
60 X 80 = 4800 bytes
```

بالمقارنة مع طابعة ليزرية تطبع ثلاثمائة نقطة في البوصة (DPI) (باعتبار أن مساحة المنطقة القابلة للكتابة في الصفحة هي ثماني بوصات بعشر بوصات):

```
(8 X 300) X (10 X 300) / 8 = 900000 bytes
```

العديد من الشبكات البطيئة التي تتصل بها الحواسيب الشخصية (آنذاك) لم تكن تتحمل نقل 1 ميغابايت من البيانات لطباعة صفحة واحدة على طابعة ليزرية، لذا كان من الضروري أن يظهر اختراع جديد لحل هذه المشكلة.

كان ذاك الاختراع الجديد هو لغة وصف الصفحات (PDL). لغة وصف الصفحات هي لغة برمجة تصف محتوى

الصفحة. تقول (بشكلٍ مبسط) للطابعة: "اذهبي إلى هذا الموضع، ارسمي الحرف "a" بخط Helvetica بحجم 10، اذهبي إلى ذاك الموضع..." حتى يُطبَع جميع محتوى الورقة. أول لغة وصف الصفحات هي PostScript من شركة أدوبي (Adobe)، التي ما زالت واسعة الانتشار إلى يومنا هذا. لغة PostScript هي لغة برمجة كاملة موجهة للطباعة وللأنواع الأخرى من الرسوميات والتصوير. تحتوي على دعم مدمج لخمس وثلاثين خطًا معياريًا عالي الدقة. بالإضافة إلى إمكانية قبول المزيد من الخطوط في زمن التنفيذ. كان دعم PostScript -في بادئ الأمر- مدمجًا بالطابعات. وهذا ما حلّ مشكلة نقل البيانات.

تقبل طابعات PostScript برنامج PostScript كمُدخل. تحتوي الطابعة على معالج وذاكرة خاصين بها وتُنفِّذ برنامجًا خاصًا يُسمى "مُفسّر PostSctipt" (PostScript interpreter)، الذي يقرأ برنامج PostScript الممرر إلى الطابعة ويحفظ الناتج في ذاكرة الطابعة الداخلية، وهذا ما يُشكِّل نمط من البتات (النقاط) التي ستطبع على الورقة. الاسم العام لعملية نقل شيء ما إلى نمط بتات أكبر منه (يسمى bitmap) هو "معالج الصور النقطية" (raster image processor أو RIP).

وبعد مضي السنوات، أصبحت الحواسيب والشبكات أسرع بكثير؛ مما مكّن عملية معالجة الصور النقطية من الانتقال من الطابعة إلى الحاسوب المضيف، الأمر الذي سمح بانخفاض سعر الطابعات ذات الجودة العالية.

ما زالت العديد من الطابعات اليوم تقبل السلاسل المحرفية كمدخلات، لكن الطابعات ذات السعر المتدني لا تدعمها. تعتمد تلك الطابعات على معالجة الصور النقطية في الحاسوب المضيف لكي تُنشئ سلسلة من البتات لطباعتها كنقاط. ما تزال طابعات PostScript موجودةً إلى الآن.

الطباعة في لينُكس

أنظمة لينُكس الحديثة تستخدم نوعين من البرمجيات للطباعة وإدارتها. أول هذين النوعين هو CUPS (النظام الشائع للطباعة في يونكس أو "Common Unix Printing System")، الذي يوفر تعاريف الطابعات وإدارة مهام الطباعة. والثاني، Ghostscript، الذي يعمل كمفسر PostScript، ويلعب دور معالج الصور النقطية (RIP).

يدير CUPS الطابعات، وذلك بإنشاء وإدارة طلبيات الطباعة. وكما ناقشنا في درس التاريخ السابق، لقد أُنشئ نظام الطباعة في يونكس لإدارة الطابعات المركزية التي يتشارك فيها أكثر من مستخدم. ولأن الطابعات بطيئة بالمقارنة مع الحواسيب، فيجب على أنظمة الطباعة ترتيب مهام الطباعة المختلفة. يملك CUPS القدرة على تمييز أنواع مختلفة من البيانات وتحويل الملفات إلى نمط قابل للطباعة.

تحضير الملفات للطباعة

نحن، كمستخدمي سطر الأوامر، مهتمين بطباعة النصوص؛ على الرغم من إمكانية طباعة العديد من أنواع البيانات المختلفة أيضًا.

تحويل الملفات النصية للطباعة باستخدام pr

لقد ألقينا نظرة سريعة على pr في الفصل السابق. الآن يمكن أن نتفحص العديد من خياراته التي تُستخدم مع الطباعة. في الموجز الذي ذكرناه عن تاريخ الطباعة، رأينا أن الطابعات التي تعتمد على الحروف تستخدم خطوطًا ذات عرضٍ ثابت، مما يؤدي إلى وجود عدد ثابت من الحروف في السطر وعدد ثابت من الأسطر في الورقة. يُستخدم pr لتعديل النص كي يتسع في قياس ورقة محدد، مع هوامش وترويسة وتذييل اختياريين. هذا ملخص عن أكثر خيارات pr استخدامًا:

الجدول 22-1: خيارات pr الشائعة

الخيار	الشرح
`+first[:last]`	طباعة مجال الصفحات التي تبدأ من `first` وتنتهي اختياريًا بالصفحة `last`.
`-columns`	تنظيم محتوى الصفحة على شكل أعمدة يُحدَّد عددها بالقيمة `columns`.
`-a`	يُعرَض المحتوى المقسّم إلى عدّة أعمدة افتراضيًا بشكل رأسي؛ نستخدم الخيار `-a` لعرضه بشكل أفقي.
`-d`	مضاعفة الفراغات في المخرجات.
`-D "format"`	تنسيق التاريخ المعروض في ترويسة الصفحة بالشكل `format`. راجع صفحة الدليل للأمر date لشرح عن عبارة التنسيق.
`-f`	استخدام محرف "form feed" بدلًا من محرف العودة إلى بداية السطر لفصل الصفحات.
`-h "header"`	استخدم السلسلة النصية "header" بدلًا من اسم الملف في منتصف الترويسة.
`-l lenght`	تحديد طول الصفحة إلى القيمة `lenght`. القيمة الافتراضية هي 66.
`-n`	عدد الأسطر.
`-o offset`	إنشاء محاذاة من اليسار عرضها هو `offset` من الحروف.
`-w width`	تحديد عرض الصفحة إلى القيمة `width`. القيمة الافتراضية هي 72.

يُستخدم pr عادةً في الأنابيب كمُرَشِّح. في هذا المثال، سننسّق محتوى المجلد `/usr/bin` على شكل ثلاثة أعمدة باستخدام الأمر pr:

```
[me@linuxbox ~]$ ls /usr/bin | pr -3 -w 65 | head

2009-02-18 14:00                                        Page 1


[                    apturl                bsd-write
411toppm             ar                    bsh
a2p                  arecord               btcflash
a2ps                 arecordmidi           bug-buddy
a2ps-lpr-wrapper     ark                   buildhash
```

إرسال مهام الطباعة إلى الطابعة

يدعم CUPS طريقتَين للطباعة أُستخدمتا في الأنظمة الشبيهة بِيونكس. إحدى الطرق تدعى Berkeley أو LPD (تُستخدم في نسخة Berkeley من يونكس)، التي تستخدم البرنامج lpr. بينما تستخدم الطريقة الثانية التي تدعى SysV (من نسخة System V من يونكس) برنامج lp. كلتا الطريقتين تقومان بنفس العمل تقريبًا. تفضيل واحدة على أُخرى هو مجرد اختيار شخصي.

طباعة الملفات باستخدام lpr

يُستخدم برنامج lpr لإرسال الملفات إلى الطابعة. يمكن أن يُستخدم في الأنابيب، حيث يقبل المدخلات من مجرى الدخل القياسي. على سبيل المثال، لطباعة ناتج تنسيق محتويات المجلد usr/bin/ في المثال الموجود في الفقرة السابقة، فإننا نستخدم الأمر الآتي:

```
[me@linuxbox ~]$ ls /usr/bin | pr -3 | lpr
```

سيُرسَل التقرير للطباعة في طابعة النظام الافتراضية. للإرسال إلى طابعة أخرى، استخدم الخيار P- كالآتي:

```
lpr -P printer_name
```

حيث printer_name هو اسم الطابعة المنشودة. استخدم الأمر الآتي لمشاهدة قائمة بأسماء الطابعات الموجودة في النظام:

```
[me@linuxbox ~]$ lpstat -a
```

تلميحة: العديد من توزيعات لينُكس تسمح لك بتعريف "طابعة" تُخرِج الملفات بصيغة PDF (صيغة المستندات المحمولة) بدل طباعتها إلى طابعة حقيقية. مما يسهل التجربة مع أوامر الطباعة. تَفَقّد برنامج إعداد الطابعات لديك لكي تعرف إذا كانت هذه الطابعة مُعرَّفة على جهازك. في بعض التوزيعات، قد تحتاج إلى تثبيت حزم إضافية (كالحزمة cups-pdf) لتفعيل هذه الميزة.

هذه قائمة ببعض الخيارات الشائعة المستخدمة مع lpr:

الجدول 22-2: خيارات lpr الشائعة

الخيار	الشرح
`-# number`	تحديد number كعدد النسخ التي ستُطبَع.
`-p`	طباعة ترويسة صغيرة تحتوي على التاريخ، والوقت، واسم المهمة، ورقم الصفحة في كل صفحة من الصفحات التي ستُطبَع.
`-P printer`	تحديد اسم الطابعة التي ستُستخدم. ستُستخدم طابعة النظام الافتراضية إذا لم يحدد هذا الخيار.
`-r`	حذف الملفات بعد الطباعة. قد يكون هذا الخيار مفيدًا للبرامج التي تنتج ملفات طباعة مؤقتة.

طباعة الملف باستخدام lp

كما في lpr، يقبل lp المدخلات كملفات أو عبر مجرى الدخل القياسي. إنه يختلف عن lpr بدعمه لمجموعة خيارات أوسع (وأكثر تعقيدًا). هذه قائمة بأشهر تلك الخيارات:

الجدول 22-3: خيارات lp الشائعة

الخيار	الشرح
`-d printer`	تحديد اسم الطابعة التي ستُستخدم. ستُستخدم طابعة النظام الافتراضية إذا لم يحدد هذا الخيار.
`-n number`	تكرار النسخ التي ستُطبَع بعدد number.
`-o landscape`	تحديد اتجاه الصفحة (رأسية أو أفقية).
`-o fitplot`	تغيير أبعاد الملف كي يتسع في الصفحة. هذا الخيار مفيد عند طباعة

	الصور، كملفات JPEG.
`-o scaling=number`	تغيير أبعاد الملف إلى number. تؤدي القيمة 100 إلى ملء الصفحة. القيم الأكبر من 100 تؤدي إلى طباعة الملف على عدّة ورق. القيم الأقل من 100 تؤدي إلى تصغير الأبعاد.
`-o cpi=number`	تحديد عدد الحروف التي ستُطبع في البوصة إلى القيمة number. القيمة الافتراضية هي 10.
`-o lpi=number`	تحديد عدد الأسطر التي ستُطبع في البوصة إلى القيمة number. القيمة افتراضية هي 6.
`-o page-bottom=points` `-o page-left=points` `-o page-right=points` `-o page-top=points`	تحديد قيم هوامش الصفحة. تُحسَب القيم بواحدة "النقطة" (point)، التي هي واحدة قياس تُستخدم في القياسات المطبعية. يوجد 72 نقطة في البوصة.
`-P pages`	تحديد قائمة بالصفحات التي ستُطبع. يمكن التعبير عن القيمة pages عن طريق قائمة بالصفحات مفصولةً فيما بينها بفاصلة و/أو مجال. على سبيل المثال "1,3,5,7-10".

سنطبع محتويات المجلد usr/bin/ مرّة أُخرى، لكن هذه المرة باستخدام اثنيّ عشر حرفًا في البوصة (CPI) وثمانية أسطر في البوصة (LPI) مع هامش أيسر بمقدار نصف بوصة. لاحظ كيف استخدمنا خيارات البرنامج pr لتحديد الأبعاد الجديدة للصفحة:

```
[me@linuxbox ~]$ ls /usr/bin | pr -4 -w 90 -l 88 | lp -o page-left=36
-o cpi=12 -o lpi=8
```

ينتج هذا الأنبوب قائمة بأربعة أعمدة تستخدم خط أصغر من الخط الافتراضي. زيادة عدد الحروف في البوصة تسمح لنا بتوسيع أعمدة أكثر في الصفحة.

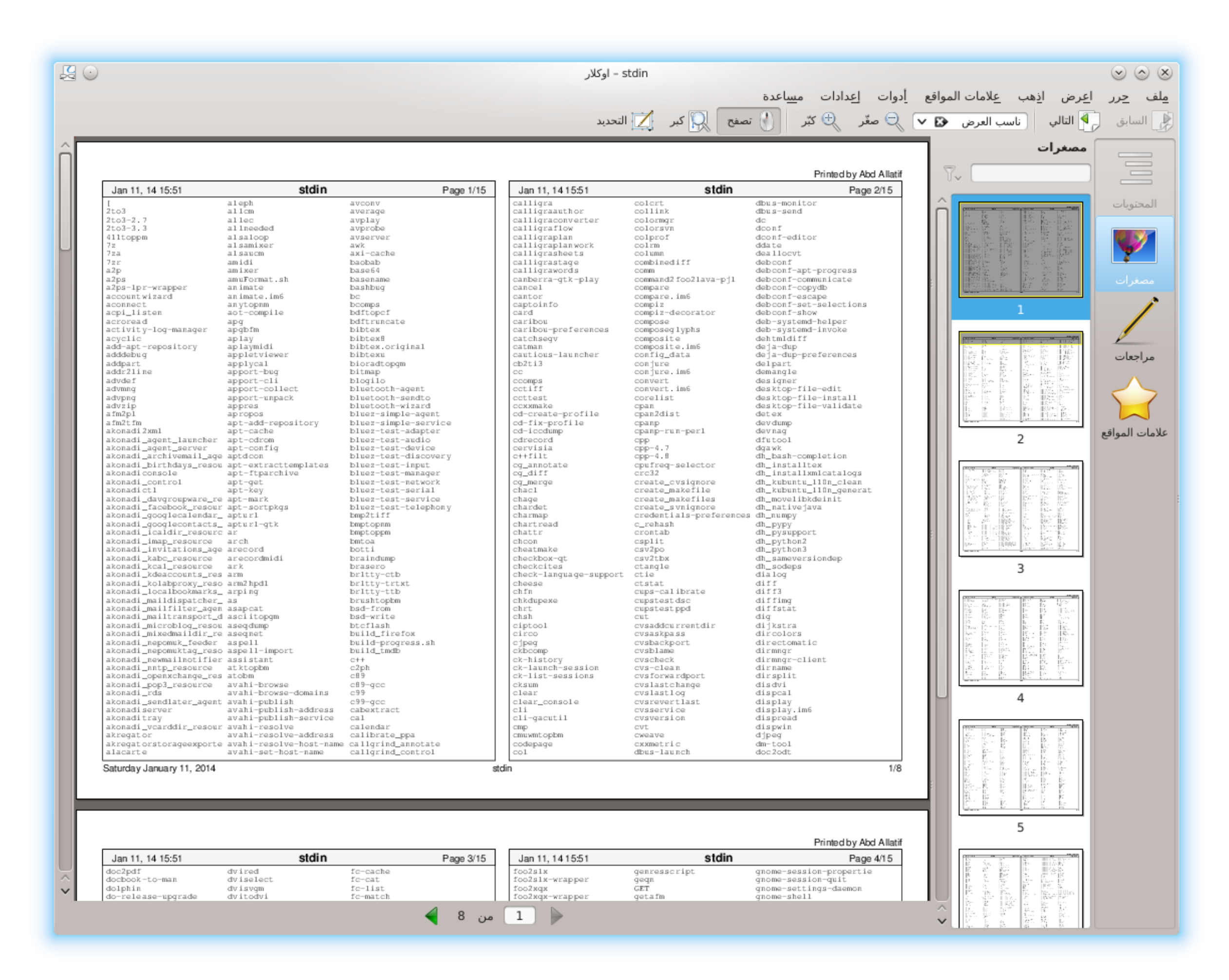
الخيار الآخر: a2ps

إن برنامج a2ps هو برنامج مثير للاهتمام. وكما هو واضح من اسمه، فهو برنامج لتحويل الصيغ، لكن هذا البرنامج يقوم بأكثر من مجرد التحويل ما بين الصيغ. يُقصد باسمه العبارة "ASCII to PostScript" ويستخدم لتحضير النصوص للطباعة في طابعات PostScript. طُوِّرت إمكانيات هذا البرنامج تطويرًا كبيرًا عبر السنوات،

وأصبح الآن يُسمى "Anything to PostScript". وعلى الرغم من أن اسمه يشير إلى أنه برنامج تحويل، إلا أنه في الواقع برنامج للطباعة. يُرسِل المخرجات إلى طابعة النظام الافتراضية بدلًا من مجرى الخرج القياسي. يقوم السلوك الافتراضي للبرنامج a2ps بتحسين مظهر المخرجات. إذا أردنا استخدام البرنامج لإنشاء ملف PostScript على سطح المكتب الخاص بنا:

```
[me@linuxbox ~]$ ls /usr/bin | pr -3 -t | a2ps -o ~/Desktop/ls.ps -L 66
[stdin (plain): 15 pages on 8 sheets]
[Total: 15 pages on 8 sheets] saved into the file
`/home/me/Desktop/ls.ps'
```

قمنا بترشيح الأنبوب باستخدام pr، مستخدمين الخيار t- (حذف الترويسات والتذييلات)، واستخدمنا بعدها a2ps، محددين اسم الملف الناتج (الخيار o-) و 66 سطرًا في الصفحة (الخيار L-) لكي نطابق مخرجات الأمر pr. إذا عاينا محتويات الملف الناتج بعارض المستندات، فإننا سنشاهد النتيجة الآتية:



الشكل 6: عرض مخرجات الأمر a2ps

كما لاحظنا، إن التخطيط الافتراضي الذي يستخدمه a2ps هو طباعة صفحتين في كل قطعة ورق. ويضيف a2ps ترويسة وتذييل جميلَين.

لدى a2ps العديد من الخيارات. الجدول الآتي يخلصهم:

الجدول 22-4: خيارات a2ps

الخيار	الشرح
`--center-title text`	تحديد قيمة العنوان الذي يتوسط الصفحة إلى القيمة "text".
`--columns number`	طباعة الصفحات كأعمدة يحدد عددها بالرقم number. القيمة الافتراضية هي 2.
`--footer text`	تحديد قيمة تذييل الصفحة إلى "text".
`--guess`	التبليغ عن أنواع الملفات الممررة كوسائط. لأن a2ps سيحاول أن يحوّل جميع البيانات، فإن هذا الخيار مفيد للتنبؤ بما سيقوم به a2ps عند تمرير ملف معين إليه.
`--left-footer text`	تحديد قيمة تذييل الصفحة اليسرى إلى القيمة "text".
`--left-title text`	تحديد قيمة عنوان الصفحة اليسرى إلى القيمة "text".
`--line-numbers=interval`	ترقيم الأسطر كل interval سطر.
`--list=defaults`	عرض الإعدادات الافتراضية.
`--list=topic`	عرض الخيارات للموضوع "topic" حيث topic هو واحد من الخيارات الآتية: "delegations" (البرامج الخارجية التي تُستخدم لتحويل البيانات)، "encodings" (الترميز)، "features" (الميزات)، "variables" (المتغيرات)، "media" (قياس الصفحة وما شابه ذلك)، "ppd" (خصائص طابعة PostScript)، "printers" (الطابعات)، "prologues" (أجزاء الكود التي ستسبق المخرجات بشكل طبيعي)، "stylesheets" (الأنماط المستخدمة)، وخيارات المستخدم.
`-pages range`	طباعة الصفحات الموجود في المجال range.

--right-footer text	تحديد قيمة تذييل الصفحة اليمنى إلى القيمة "text".
--right-title text	تحديد قيمة عنوان الصفحة اليمنى إلى القيمة "text".
--rows number	طباعة الملفات كصفوف يحدد عددها بالرقم number. القيمة الافتراضية هي 1.
-B	عدم إظهار ترويسة الصفحة.
-b text	تحديد قيمة نص الترويسة إلى "text".
-f size	استخدم حجم الخط size.
-l number	تحديد قيمة عدد المحارف في السطر إلى number. يمكن استخدام هذا الخيار والخيار L- لتحويل الملفات إلى صفحات باستخدام برامج أُخرى كبرنامج pr.
-L number	تحديد قيمة عدد الأسطر في الصفحة إلى number.
-M name	تحديد قياس الصفحة، على سبيل المثال "A4".
-n number	طباعة كل صفحة بعدد number.
-o file	إرسال المخرجات إلى الملف file. إذا حُدِدَ الرمز "-"، فسيستخدم مجرى الخرج القياسي.
-P printer	استخدام الطابعة printer. إذا لم يتم تحديد هذا الخيار، فسيُطبَع باستخدام طابعة النظام الافتراضية.
-R	جعل اتجاه الصفحة طوليًا.
-r	جعل اتجاه الصفحة عرضيًا.
-u text	إضافة علامة مائية (watermark) إلى الصفحات تحتوي على النص "text".

الجدول السابق ملخص لبعض خيارات a2ps فقط. لدى a2ps العديد من الخيارات.

ملاحظة: ما يزال برنامج a2ps قيد التطوير. لاحظت، أثناء تجربتي له، أنه يبدي سلوكًا مختلفًا في مختلف التوزيعات. ستذهب المخرجات إلى مجرى الخرج القياسي افتراضيًا في توزيعة CentOS 4. وقياس الورق الافتراضي المُستعمل في توزيعتَيّ فيدورا و CentOS هو A4، على الرغم من أن البرنامج قد ضُبِطَ كي يستخدم قياس أوراق مختلف (letter). استطعت التغلب على هذه الإشكاليات الصغيرة بتحديد بعض الخيارات. يجدر بالذكر أن a2ps يُنَفَّذ كما هو موجود في توثيقه في توزيعة أوبنتو.

لاحظ أيضًا أن هنالك برنامجًا آخر لتنسيق المخرجات يفيد في تحويل النصوص إلى صيغة PostScript يُدعى `enscript`؛ يمكن أن يستعمل هذا البرنامج لتطبيق مختلف أنواع التنسيقات. لكنه لا يدعم سوى المدخلات النصية (على عكس a2ps).

مراقبة والتحكم في مهام الطباعة

لما كانت أنظمة الطباعة في يونكس قد صُمِّمت لمعالجة أكثر من مهمة طباعة من أكثر من مستخدم، وكذلك صُمِّم CUPS لأداء نفس العمل. تُعطى كل طابعة "طابور الطباعة" (print queue)، حيث تُخزَّن مهام الطباعة فيه قبل أن تُرسَل إلى الطابعة لطباعتها. يوفر CUPS عدّة برامج لسطر الأوامر للتحكم في حالة الطابعة وطوابير الطباعة. وكما في برنامجي `lpr` و `pr`، بُنِيَت أدوات الإدارة مشابهةً لنظيراتها في أنظمة الطباعة المستخدمة في Berkeley و System V.

عرض حالة منظومة الطباعة باستخدام lpstat

يستخدم برنامج `lpstat` لتحديد أسماء وتوفر الطابعات في النظام. على سبيل المثال، إذا كان لدينا نظام يحتوي على طابعتَين، إحداهما فيزيائية (تسمى "`printer`")، والأخرى طابعة PDF وهمية (مسماة "PDF")، فيكون أمر التأكد من حالتهما:

```
[me@linuxbox ~]$ lpstat -a
PDF accepting requests since Mon 08 Dec 2008 03:05:59 PM EST
printer accepting requests since Tue 24 Feb 2009 08:43:22 AM EST
```

يمكننا أيضًا الحصول على مزيدٍ من التفاصيل حول طريقة إعداد نظام الطباعة بالطريقة الآتية:

```
[me@linuxbox ~]$ lpstat -s
system default destination: printer
device for PDF: cups-pdf:/
device for printer: ipp://print-server:631/printers/printer
```

يمكننا ملاحظة أن الطابعة "printer" هي طابعة النظام الافتراضية وهي موصولة عبر الشبكة باستخدام بروتوكول الطباعة عن بعد (ipp://)، وهي موصولة إلى نظام يسمى "print-server".

هذه قائمة بأكثر الخيارات فائدةً:

الجدول 22-5: خيارات lpstat الشائعة

الخيار	الشرح
-a [printer...]	عرض حالة طابور الطباعة للطابعة printer. لاحظ أن هذا الخيار سيُظهِر حالة طابور الطباعة وفيما إذا كان يستطيع قبول مهام طباعة جديدة، وليس حالة الطابعة الفيزيائية. ستُعرَض معلومات عن جميع الطابعات إذا لم يحدد اسم الطابعة.
-d	عرض اسم طابعة النظام الافتراضية.
-p [printer...]	عرض حالة الطابعة printer. ستُعرَض حالة جميع الطابعات إذا لم يحدد اسم الطابعة.
-r	عرض حالة خادم الطباعة.
-s	عرض مخلص عن الحالة.
-t	عرض تقرير مُفَصَّل عن الحالة.

إظهار طابور الطباعة باستخدام lpq

يُستخدم برنامج lpd لعرض حالة طابور الطباعة. يسمح لنا هذا البرنامج بمعرفة حالة المهام التي يحتويها الطابور. هذا مثال عن طابور طباعة فارغ لطابعة النظام الافتراضية:

```
[me@linuxbox ~]$ lpq
printer is ready
no entries
```

إذا لم نحدد اسم الطابعة (باستخدام الخيار P-)، ستُظهَر حالة الطابعة الافتراضية للنظام. إذا أرسلنا مهمة طباعة إلى الطابعة ثم فحصنا بعدها الطابور، فسوف نشاهد المهمة في القائمة:

```
[me@linuxbox ~]$ ls *.txt | pr -3 | lp
```

```
request id is printer-603 (1 file(s))
[me@linuxbox ~]$ lpq
printer is ready and printing
Rank       Owner     Job       File(s)                Total Size
active     me        603       (stdin)                1024 bytes
```

إلغاء مهام الطباعة باستخدام lprm/cancel

يوفر CUPS برنامجَين يُستخدمان لإلغاء مهام الطباعة وإزالتهم من الطابور. أحد هذان البرنامجان هو `lprm` (نمط Berkeley) والآخر هو `cancel` (نمط System V). يختلفان فيما بينهما بالخيارات التي يدعمانها، لكنهما يقومان بنفس العمل تقريبًا. يمكننا إيقاف مهمة الطباعة التي أنشأناها في المثال السابق بالأمر الآتي:

```
[me@linuxbox ~]$ cancel 603
[me@linuxbox ~]$ lpq
printer is ready
no entries
```

لدى كل أمرٍ منهما خياراتٌ لإزالة جميع المهام التي تتعلق بمستخدم معين، أو طابعة معينة، أو بتحديد أرقام المهام. راجع صفحة الدليل لكل أمر للحصول على تفاصيلٍ أوفر.

الخلاصة

لقد رأينا في هذا الفصل كيف أثرت طابعات "الماضي" على تصميم نظم الطباعة في الأنظمة الشبيهة بِيونكس، ومقدار التحكم الكبير الذي يوفره سطر الأوامر، ليس فقط للطباعة وإدارة المهام، بل وتوفير عدّة خيارات لتنسيق المخرجات.

الفصل الثالث والعشرون:

بناء البرامج

سنلقي في هذا الفصل نظرةً على طريقة بناء البرامج بتصريف الكود المصدري. توافر الكود المصدر للبرامج هو حرية أساسية لم يكن نظام لينُكس موجودًا دونها. تعتمد فلسفة لينُكس على التبادل الحر للبرمجيات بين المطورين. كان مستخدمو سطح المكتب العاديون يبنون التطبيقات التي يستخدمونها، لكن في هذه الأيام، تحتوي مستودعات التوزيعات على الكثير من البرامج المبنية والجاهزة للاستخدام. تحتوي مستودعات أوبنتو -في وقت كتابة هذا الكتاب- على أكثر من أربعين ألف حزمة.

إذًا، لماذا نبني البرمجيات يدويًا؟ يوجد سببان لذلك:

1. **التوفر.** على الرغم من الرقم الكبير للحزم المبنية في المستودعات، لكن بعض التوزيعات لا تحتوي على كل البرمجيات المطلوبة. في هذه الحالة، الطريقة الوحيدة للحصول على برنامجٍ ما غير متوفر في المستودعات هي بناؤه من المصدر.
2. **الحصول على آخر إصدارات البرامج.** بينما تتخصص بعض التوزيعات بتوفيرها لآخر الإصدارات من البرامج، إلا أن بعضها الآخر لا يقوم بذلك. هذا يعني أن الطريقة الوحيدة للحصول على آخر إصدارات برنامجٍ ما هي بناؤه من المصدر.

يمكن أن تصبح عملية تصريف البرامج من المصدر معقدةً جدًا؛ ولا يمكن لأغلب المستخدمين القيام بها. لكن العديد من مهام البناء هي مهمات سهلة للغاية وتقتضي القيام ببعض الخطوات البسيطة. ذاك الأمر يعتمد على الحزمة. سنلقي نظرةً على حالة بسيطة كي نأخذ فكرة عن البناء من المصدر؛ ولتشكيل نقطة انطلاق للأشخاص الذين يريدون الإبحار في هذا المجال.

سنستخدم أمرًا جديدًا وحيدًا:

- `make` – أداة تستخدم في عملية تصريف البرامج من المصدر.

ما هو التصريف؟

ببساطة، إن التصريف (compiling) هو عملية تحويل الكود المصدري (source code [الجزء القابل للقراءة من قِبل البشر من البرنامج الذي يكتبه المبرمجون]) إلى لغة معالج الحاسوب.

يُنفِّذ معالج الحاسوب (CPU) البرامج في ما يسمى لغة الآلة (machine language). التي هي عبارة عن أكواد عددية تصف أوامر صغيرة جدًا، كالأمر "أضف هذا البايت"، أو "أشر إلى هذا الموضع في الذاكرة"، أو

"انسخ هذا البايت". يُعبَّر عن هذه الأوامر بالأصفار والواحدات (النظام الثنائي). كانت البرامج الأولى في الحاسوب تُكتَب بهذه الطريقة، وهذا ما يفسر شرب المبرمجين الذين كتبوها للكثير من القهوة كل يوم، بالإضافة إلى معاناتهم من ضعف النظر.

حُلّت هذه الإشكالية بإنشاء لغة التجميع (assembly language)، التي استبدلت الأوامر الرقمية بأكواد حرفية يسهل حفظها كالأمر CPY (للنسخ) و MOV (للنقل). تُعالَج البرامج المكتوبة بلغة التجميع وتُحوَّل إلى لغة الآلة باستخدام برنامج يسمى "المُجَمِّع" (assembler). ما تزال لغة التجميع مستخدمةً إلى يومنا هذا لكتابة بعض المهام البرمجية الخاصة، كتعاريف الأجهزة والنظم المدمجة (embedded systems).

تلا ذلك إنشاء لغات البرمجة عالية المستوى. أُطلق هذا الاسم عليها لأنهم يسمحون للمبرمج بالتركيز على المشكلة التي يريد حلّها دون الاهتمام بتفاصيل الأوامر التي ينفذها المعالج. كانت أولى تلك اللغات هي Fortran (صُمِّمَت للقيام بالمهام العلمية والتقنية) و COBOL (صُمِّمَت للتطبيقات التجارية). لا تُستخدَم هاتان اللغتان حاليًا إلا ببعض الاستخدامات المحدودة.

وعلى الرغم من ظهور عدد من لغات البرمجة عالية المستوى التي اشتهرت فيما بعد بين المبرمجين، لكن لغتين أثبتتا تفوقهما، وتُكتَب أغلب البرامج للأنظمة الحديثة بهما، إنهما C و ++C. سنحاول بناء برنامج مكتوب بلغة C في الأمثلة القادمة.

تحول البرامج التي كُتِبت بلغة عالية المستوى إلى لغة الآلة باستخدام برنامج آخر اسمه "المُصرِّف" (compiler أو كما يسميه البعض "المترجم"). تحول بعض المصرّفات أوامر اللغة عالية المستوى إلى لغة التجميع ومن ثم تستخدم مُجمِّع لتحويلها إلى لغة الآلة.

عملية أخرى ترافق عملية التصريف هي عملية الربط (linking). يوجد هنالك العديد من المهام الشائعة التي تُنفِّذها البرامج. على سبيل المثال، فتح الملفات. تقوم برامج عديدة بهذه المهمة، لكن ليس عمليًّا أن يُنشِئ كل برنامج آلية فتح ملفات خاصة به. من المفيد أن تُنشَأ قطعة من الأكواد تحوى على آلية فتح الملفات وتسمح باستخدامها لجميع البرامج التي تحتاجها. هذا يتم باستخدام ما يُعرَف بالمكتبات (libraries). التي تحتوي على عدّة صيغ مكتبية (routines)، تقوم كلٌ منها بمهمة معينة يتشارك فيها عدّة برامج. توجد هذه المكتبات عادةً في مجلدَيّ `/lib` و `/usr/lib`. يُستخدم برنامج يسمى "الموصل" (linker) للربط ما بين ناتج عملية التصريف والمكتبات التي يحتاج إليها البرنامج الذي صُرِّفَ. النتيجة النهائية لهذه العملية هي ملف تنفيذي للبرنامج جاهز للاستخدام.

هل جميع البرامج مُصرّفة؟

لا. توجد العديد من البرامج كسكربتات الشِل التي لا تحتاج إلى تصريف. تُنفَّذ هذه البرامج مباشرةً. تُكتَب هذه البرامج بلغات تسمى اللغات المفسرة (interpreted languages) أو لغات السكربتات (scripting languages). انتشرت هذه اللغات انتشارًا كبيرًا في السنوات الأخيرة، ينضوي تحت هذا اللواء لغات Perl، و Python،

و PHP، و Ruby وغيرها الكثير.

تُنفَّذ لغات السكربتات باستخدام برنامج خاص يسمى "المفسر" (interpreter). يقرأ المفسر كل تعليمة في ملف البرنامج ويُنفِّذها. عمومًا، تكون البرامج المفسرة أبطأ بالتنفيذ بالمقارنة مع البرامج المصرفة. وذلك لأن البرنامج المفسر سيُحوَّل إلى لغة الآلة في كل مرّة يُنفَّذ فيها على عكس البرنامج المصرف الذي يُحوَّل لمرّة واحدة فقط إلى لغة الآلة.

لماذا إذًا اللغات المفسرة مشهورة إلى هذا الحد؟ لأن تنفيذ البرامج المفسرة "سريع سرعةً كافيةً" لأغلب المهام البرمجية الاعتيادية، لكن الميزة الحقيقية هي أن تطوير البرامج المفسرة أسرع بكثير من البرامج المصرفة. تُطوَّر البرامج عادةً بتناوب المراحل "كتابة البرنامج" ثم "تصريف البرنامج" ثم "التجربة". وعندما يكبر البرنامج في الحجم، ستستغرق مرحلة التصريف وقتًا طويلًا. توفر اللغات المفسرة وقت التصريف مما يؤدي إلى تسريع التطوير بها.

بناء برنامج مكتوب بلغة C

لنحاول الآن بناء برنامج ما. لكن قبل أن نبدأ عملية البناء؛ سنحتاج إلى بعض الأدوات كالمصرف، والموصل (linker)، والأداة make. مصرف C الذي يستخدم في جميع أنظمة لينُكس تقريبًا هو gcc (GNU C Compiler)، الذي كتبه ريتشارد ستالمان. لا تُضمّن أغلب توزيعات لينُكس gcc افتراضيًا. يمكننا التحقق من توفر المصرف من عدمه في نظامنا بالأمر الآتي:

```
[me@linuxbox ~]$ which gcc
/usr/bin/gcc
```

تُشير النتائج في المثال السابق إلى وجود المصرف على نظامنا.

تلميحة: قد تحتوي توزيعتك على حزمة وصفية (meta-package أي مجموعة من الحزم) تحتوي على البرامج الضرورية لبناء البرمجيات. ثبّتها -إن كانت متوفرة- على نظامك إذا أردت بناء البرامج على جهازك. إن لم توفر توزيعتك تلك الحزمة، فجرّب تثبيت حزمتي gcc و make لأنهما كفيلتان (في أغلب التوزيعات) بتنفيذ التمارين الموجودة في هذا الفصل.

الحصول على الكود المصدري

سنبني برنامجًا من مشروع غنو يُسمى diction. الذي هو برنامج صغير ومفيد للتحقق من جودة ونمط الكتابة المستخدم في الملفات النصية. قد تم اختيار هذا البرنامج لأنه صغير وسهل البناء.

سنتبع أعراف بناء البرامج، وننشئ مجلدًا يحتوي على الكود المصدري باسم src، ومن ثم سنُنزّل الكود المصدري إلى ذاك المجلد باستخدام عميل ftp:

```
[me@linuxbox ~]$ mkdir src
[me@linuxbox ~]$ cd src
[me@linuxbox src]$ ftp ftp.gnu.org
Connected to ftp.gnu.org.
220 GNU FTP server ready.
Name (ftp.gnu.org:me): anonymous
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd gnu/diction
250 Directory successfully changed.
ftp> ls
 200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--       1 1003 65534   68940 Aug 28 1998 diction-0.7.tar.gz
-rw-r--r--       1 1003 65534   90957 Mar 04 2002 diction-1.02.tar.gz
-rw-r--r--       1 1003 65534  141062 Sep 17 2007 diction-1.11.tar.gz
226 Directory send OK.
ftp> get diction-1.11.tar.gz
local: diction-1.11.tar.gz remote: diction-1.11.tar.gz
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for diction-1.11.tar.gz
(141062 bytes).
226 File send OK.
141062 bytes received in 0.16 secs (847.4 kB/s)
ftp> bye
221 Goodbye.
[me@linuxbox src]$ ls
diction-1.11.tar.gz
```

ملاحظة: لما كنّا نحن "المسؤولون" عن الكود المصدري عند بناءه، فإننا سنضع الكود المصدري في مجلد src. تُخزَّن الأكواد المصدرية المُثبّتة من قِبل صانعي التوزيعة في مجلد /usr/src. بينما الأكواد

المصدرية التي قد يستخدمها عدّة مستخدمين فإنها تتواجد عادةً في مجلد usr/local/src/.

كما لاحظنا، يتم توفير الأكواد المصدرية على شكل أرشيفات tar مضغوطة، تحتوي هذه الأرشيفات على "شجرة الأكواد" (source tree). أي المجلدات والملفات التي تُشَكِّل الكود المصدري. بعد اتصالنا بخادم FTP، تفحصنا قائمة الأرشيفات الموجودة واخترنا أحدث إصدار لكي نُنَزّله باستخدام الأمر get داخل عميل FTP.

وبعد أن ينتهي تنزيل الأرشيف المضغوط، نستطيع استخراج محتوياته وذلك باستخدام برنامج tar:

```
[me@linuxbox src]$ tar xzf diction-1.11.tar.gz
[me@linuxbox src]$ ls
diction-1.11  diction-1.11.tar.gz
```

تلميحة: يَتّبع برنامج diction، كغيره من برامج مشروع غنو، معايير قياسية لتحزيم الأكواد المصدرية. أغلب البرامج التي تدور في فلك لينُكس والبرمجيات الحرّة تتبع هذه المعايير. أحد عناصر هذه المعايير هو آلية استخراج أرشيفات tar التي تحتوي على الكود المصدري؛ حيث سيُنشأ مجلد يحتوي على شجرة الأكواد باسم project-x.xx حيث يتألف اسمه من اسم المشروع ورقم الإصدارة. هذا ما يُمكّن تثبيت أكثر من نسخة من كل برنامج على النظام. لكن تفحص محتوى الأرشيف قبل استخراجه هو فكرة جيدة. تضع بعض البرامج شجرة الأكواد داخل مجلد العمل الحالي مباشرةً مما يؤدي إلى حدوث فوضى في مجلد src. لتجنب ذلك، نفِّذ الأمر الآتي لكي تعاين محتويات أرشيف tar:

```
tar tzvf tarfile | head
```

معاينة شجرة الأكواد

تُنتِج عملية استخراج الأرشيف السابق مجلدًا جديدًا باسم diction. يحتوي هذا المجلد على شجرة الأكواد. لنلقِ عليه نظرة:

```
[me@linuxbox src]$ cd diction-1.11
[me@linuxbox diction-1.11]$ ls
config.guess diction.c              getopt.c          nl
config.h.in       diction.pot       getopt.h          nl.po
config.sub        diction.spec      getopt_int.h      README
configure         diction.spec.in   INSTALL           sentence.c
configure.in      diction.texi.in   install-sh        sentence.h
COPYING           en                Makefile.in       style.1.in
```

```
de              en_GB              misc.c          style.c
de.po           en_GB.po           misc.h          test
diction.1.in    getopt1.c          NEWS
```

توفر البرامج التي تنتمي إلى مشروع غنو (وغيرها الكثير)، ملفات التوثيق README، و INSTALL، و NEWS، و COPYING. تحتوي هذه الملفات على شرح للبرنامج، ومعلومات عن طريقة تصريفه وتثبيته، ورخصة الاستخدام. من الجيد قراءة ملفات README و INSTALL قبل البدء في عملية بناء البرنامج.

الملفات الأخرى المثيرة للاهتمام في المجلد هي تلك التي تنتهي بالامتدادين "c." و "h.":

```
[me@linuxbox diction-1.11]$ ls *.c
diction.c getopt1.c getopt.c misc.c sentence.c
style.c
[me@linuxbox diction-1.11]$ ls *.h
getopt.h getopt_int.h misc.h sentence.h
```

تحتوي ملفات c. السابقة على برنامَجي c زُوِدا من قِبل الحزمة (style و diction)، ومُقسمَين إلى واحدات (modules). من الشائع أن تُقَسّم البرامج الكبيرة إلى قطع أقصر وأسهل من ناحية الإدارة والتطوير. ملفات الأكواد المصدرية هي ملفات نصية عادية يمكن أن نشاهد محتواها بالأمر less:

```
[me@linuxbox diction-1.11]$ less diction.c
```

الملفات ذات الامتداد h. -المعروفة بالملفات الرأسية (header files)- هي ملفات نصية أيضًا. تحتوي الملفات الرأسية على المهام البرمجية التي تُستخدم في أكواد البرنامج أو في المكتبات. ولكي يستطيع المصرف أن يربط الواحدات، يجب أن يستقبل وصفًا عن جميع الواحدات المطلوبة لإكمال كامل البرنامج. سنشاهد هذا السطر في بداية الملف diction.c:

```
#include "getopt.h"
```

السطر السابق يوجه المُصرّف إلى قراءة الملف getopt.h عندما يقرأ الملف المصدري getopt.c لكي "يعرف" ما الذي يحتويه الملف "getopt.h". يوفر الملف getopt.c المهام المشتركة ما بين برنامَجيّ style و diction.

السطر السابق يجعل المصرف يقرأ ملف getopt.h. يمكننا ملاحظة عبارات include أخرى في الملف:

```
#include <regex.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

تشير هذه العبارات أيضًا إلى الملفات الرأسية التي تتواجد خارج شجرة أكواد البرنامج الحالي. يوفر النظام هذه الملفات الرأسية لدعم إمكانية تصريف كل البرامج. إذا ألقينا نظرة على المجلد /usr/include، فسوف نشاهدها:

```
[me@linuxbox diction-1.11]$ ls /usr/include
```

أُنشئت الملفات الرأسية في ذاك المجلد عند تثبيت المُصرِّف.

بناء البرنامج

تُبنى أغلب البرامج بأمرين بسيطين فقط:

```
./configure
make
```

برنامج configure هو سكربت شِل يُوفَّر مع شجرة الأكواد. مهمته هي تحليل بيئة البناء. صُمِّمت أغلب الأكواد البرمجية لكي تكون "محمولة" (portable). ذلك يعني إمكانية بناء البرنامج على أكثر من نوع من الأنظمة الشبيهة بِيونكس دون مشاكل. لكي يتم ذلك، يجب تعديل الكود المصدري تعديلاتٍ طفيفة أثناء البناء كي يتلاءم مع الاختلافات ما بين الأنظمة. يتحقق configure أيضًا من تثبيت الأدوات والمكونات الخارجية. لننفِّذ الآن configure. ولأن الملف configure ليس موجودًا في أحد المجلدات التي تتوقع الصدفة العثور على الملفات التنفيذية فيها؛ فإننا سنضيف ./ قبل اسم الملف لكي نخبر الصدفة أن الملف التنفيذي موجود في مجلد العمل الحالي:

```
[me@linuxbox diction-1.11]$ ./configure
```

سيطبع configure مخرجات كثيرة أثناء اختباره وإعداده لعملية البناء. عندما ينتهي من التنفيذ، فسيطبع شيئًا شبيهًا بالآتي:

```
checking libintl.h presence... yes
checking for libintl.h... yes
checking for library containing gettext... none required
configure: creating ./config.status
```

```
config.status: creating Makefile
config.status: creating diction.1
config.status: creating diction.texi
config.status: creating diction.spec
config.status: creating style.1
config.status: creating test/rundiction
config.status: creating config.h
[me@linuxbox diction-1.11]$
```

المهم هنا هو عدم وجود رسائل خطأ. إذا ظهرت رسائل الخطأ، فإن عملية الإعداد ستفشل، ولن يُبنى البرنامج حتى تُصحَّح تلك الأخطاء.

لاحظنا أن configure يُنشئ عدّة ملفات جديدة في مجلد الأكواد. أهم تلك الملفات هو Makefile. ملف Makefile هو ملف إعدادات يُوجِّه make لبناء البرنامج. من دون هذا الملف، فإن make يرفض أن يُنَفَّذ. ملف Makefile هو ملف نصي عادي بإمكاننا عرض محتوياته:

```
[me@linuxbox diction-1.11]$ less Makefile
```

يأخذ البرنامج make ملف بناء كمدخلات (الذي يسمى عادةً بالاسم Makefile)؛ الذي يصف العلاقات ما بين مكونات البرنامج النهائي والاعتماديات.

يُعَرِّف القسم الأول من ملف البناء المتغيرات التي ستستخدم في أقسام أخرى من الملف. على سبيل المثال، السطر الآتي:

```
CC=       gcc
```

الذي يحدد مصرف gcc لتصريف ملفات C. نشاهد أحد الأسطر التي تستخدم المتغير السابق في مكانٍ لاحق من الملف:

```
diction:        diction.o sentence.o misc.o getopt.o getopt1.o
                $(CC) -o $@ $(LDFLAGS) diction.o sentence.o misc.o \
                getopt.o getopt1.o $(LIBS)
```

ستُستبدل $(CC) بالقيمة gcc في وقت التنفيذ.

تحدد أغلب الأسطر في ملف البناء الملف الهدف، الذي هو في هذه الحالة الملف التنفيذي diction، والملفات التي يعتمد عليهم هذا البرنامج. باقي الأسطر تحدد الأمر (أو الأوامر) الذي يُستخدم لإنشاء الملف الهدف من مكوناته. سنشاهد في هذا المثال أن الملف التنفيذي diction يتعمد على وجود diction.o،

و sentence.o، و misc.o، و getopt.o، و getopt1.o. سنشاهد في مكان لاحق من الملف تعريفات هذا الملفات كملفات هدف:

```
diction.o:      diction.c config.h getopt.h misc.h sentence.h
getopt.o:       getopt.c getopt.h getopt_int.h
getopt1.o:      getopt1.c getopt.h getopt_int.h
misc.o:         misc.c config.h misc.h
sentence.o:     sentence.c config.h misc.h sentence.h
style.o:        style.c config.h getopt.h misc.h sentence.h
```

لكن لن تشاهد أيّة أوامر محددة لأجل بنائهم. يحدد ذلك بأمر يُستخدم لتصريف أي ملف ".c" إلى ".o":

```
  .c.o:
                      $(CC) -c $(CPPFLAGS) $(CFLAGS) $<
```

يبدو ذلك معقدًا جدًا! لماذا لا تحدد ببساطة كل الخطوات اللازمة لتصريف جميع الأقسام؟ سيتضح سبب ذلك خلال لحظات. لننفذ الآن الأمر make ونبني البرنامج:

```
[me@linuxbox diction-1.11]$ make
```

سيُشغل البرنامج make مُستخدمًا محتويات الملف Makefile كدليل لما سيقوم به من أفعال. وسيطبع العديد من الرسائل.

عندما ينتهي تنفيذ make، فسنشاهد أن جميع الملفات الهدف قد أُنشئت في المجلد الحالي:

```
[me@linuxbox diction-1.11]$ ls
config.guess    de.po            en            install-sh   sentence.c
config.h        diction          en_GB         Makefile     sentence.h
config.h.in     diction.1        en_GB.mo      Makefile.in  sentence.o
config.log      diction.1.in     en_GB.po      misc.c       style
config.status   diction.c        getopt1.c     misc.h       style.1
config.sub      diction.o        getopt1.o     misc.o       style.1.in
configure       diction.pot      getopt.c      NEWS         style.c
configure.in    diction.spec     getopt.h      nl           style.o
COPYING         diction.spec.in  getopt_int.h  nl.mo        test
de              diction.texi     getopt.o      nl.po
de.mo           diction.texi.in  INSTALL       README
```

نستطيع ملاحظة الملفين diction و style بين الملفات الناتجة، هذان هما البرنامجان اللذان نريد بناءهما. تهانينا! لقد بنينا أول برنامج من المصدر!

لنجرّب الآن تنفيذ الأمر make مرّة أخرى:

```
[me@linuxbox diction-1.11]$ make
make: Nothing to be done for `all'.
```

لقد أظهر الأمر make رسالة غريبة. ما الذي يحدث؟ لماذا لم يُبنى البرنامج مرّة أُخرى؟ هذا هو الجزء الجميل في make. بدلًا من بناء كل شيء من الصفر، فسيبني make الجزء الذي يحتاج إلى بناء فقط! قرر make أنه لن يحتاج إلى القيام بأي شيء لأن جميع الملفات الهدف موجودة. لنجرب الآن حذف أحد الملفات الهدف ثم نجرّب تشغيل make مرّة أُخرى لمراقبة ما الذي سيفعله make:

```
[me@linuxbox diction-1.11]$ rm getopt.o
[me@linuxbox diction-1.11]$ make
```

لاحظنا أن make يُعيد بناء ووصل برنامجي style و diction، لأنهما يعتمدان على وحدة ناقصة. يشير هذا السلوك إلى ميزة أُخرى مهمة في make: إنه يُحدِّث الملفات الهدف بعد كل عملية بناء. يُصّر make على جعل الملفات الهدف "أجدد" من اعتمادياتها. وهذا أمر منطقي جدًا، حيث يُحدِّث المبرمج عادةً قسمًا صغيرًا من الكود المصدري ويستخدم make لبناء نسخة جديدة من البرنامج. يتأكد make من أن أي شيء يعتمد على الكود المُعدَّل سيُعاد بناؤه. إذا استخدمنا الأمر touch لتغيير بصمة الوقت لأحد ملفات الأكواد:

```
[me@linuxbox diction-1.11]$ ls -l diction getopt.c
-rwxr-xr-x 1    me     me 37164 2009-03-05 06:14    diction
-rw-r--r-- 1    me     me 33125 2007-03-30 17:45    getopt.c
[me@linuxbox diction-1.11]$ touch getopt.c
[me@linuxbox diction-1.11]$ ls -l diction getopt.c
-rwxr-xr-x 1    me     me 37164 2009-03-05 06:14    diction
-rw-r--r-- 1    me     me 33125 2009-03-05 06:23    getopt.c
[me@linuxbox diction-1.11]$ make
```

بعد تنفيذ make، سوف نلاحظ أن الملف الهدف سيكون "أحدث" من الاعتمادية:

```
[me@linuxbox diction-1.11]$ ls -l diction getopt.c
-rwxr-xr-x 1 me     me   37164 2009-03-05 06:24 diction
-rw-r--r-- 1 me     me   33125 2009-03-05 06:23 getopt.c
```

قدرة make على بناء الأجزاء التي تحتاج إلى إعادة بناء فقط تجعله أداةً نافعةً جدًا للمبرمجين. وعلى الرغم من أن توفير الوقت لم يظهر بشكلٍ جليٍ في برنامجنا الصغير، لكنه مهم جدًا في المشاريع الأكبر. تذكر أن نواة لينُكس (برنامج دائم التغيير والتطوير) تحتوي على ملايين الأسطر من الأكواد البرمجية.

تثبيت البرنامج

تحتوي الأكواد المصدرية المحزَّمة جيدًا غالبًا على ملف بناء للبرنامج make اسمه `install`. يُثبِّت هذا الملف البرنامج النهائي في النظام لكي نستخدمه. عادةً، يكون هذا المجلد هو `/usr/local/bin`، وهو المكان التقليدي للبرامج المبنية محليًا. لكن ليس للمستخدمين العاديين إذن الكتابة على ذاك المجلد، لذا، سنحتاج إلى استخدام حساب الجذر للقيام بعملية التثبيت:

```
[me@linuxbox diction-1.11]$ sudo make install
```

بعد القيام بالتثبيت، نستطيع أن نتأكد من أن البرنامج يعمل عملًا صحيحًا باستخدام:

```
[me@linuxbox diction-1.11]$ which diction
/usr/local/bin/diction
[me@linuxbox diction-1.11]$ man diction
```

وها قد أصبح مثبتًا على نظامنا!

الخلاصة

شاهدنا في هذا الفصل كيف يمكن لثلاثة أوامر بسيطة:

```
./configure
make
make install
```

أن تبني العديد من الحزم المصدرية للبرامج. وناقشنا أيضًا الدور المهم الذي يلعبه الأمر make في صيانة البرامج. يمكن للبرنامج make أن يُستخدم في أية مهمة تحتاج إلى صون العلاقة ما بين الملف الهدف والاعتمادية، وليس فقط لتصريف الأكواد.

تُرِكَتْ هَذِهِ الصَفْحَةُ فَارِغَةً عَمْدًا

الباب الرابع:
كتابة سكربتات شِل

الفصل الرابع والعشرون:
كتابة أول سكربت لك

لقد جمعنا في رحلتنا الطويلة في الفصول الماضية ترسانةً من أدوات سطر الأوامر. وعلى الرغم من أن تلك الأدوات تستطيع حلّ العديد من المشاكل التي قد تواجهك، لكنها محدودة، حيث ستحتاج إلى إدخالها يدويًا سطرًا بسطر في الطرفية. أليس من الجيد أن نجعل الصدفة تقوم بأغلب هذه الأعمال؟ حسنًا، نستطيع ذلك بجمع الأدوات مع بعضها البعض وتشكيل برامج نصممها نحن، ومن ثم تُنفّذها الصدفة. يمكننا فعل ذلك بكتابة سكربتات شِل (shell scripts).

ما هي سكربتات الشِل؟

ببساطة، سكربت الشِل هو ملف يحتوي على سلسلة من الأوامر. تقرأ الصدفة الملف وتنفذ الأوامر الموجودة فيه كما لو أُدخلت مباشرةً من سطر الأوامر.

الصدفة هي برنامج مميز بعض الشيء، يمكن استخدامها كواجهة للتعامل مع النظام وكمفسّر للسكربتات. كما سنشاهد لاحقًا، أغلب الأشياء التي نستطيع فعلها في سطر الأوامر نستطيع فعلها أيضًا في السكربتات؛ والعكس صحيح، أغلب الأشياء التي نستطيع فعلها في السكربتات نستطيع فعلها في سطر الأوامر.

لقد تعلمنا الكثير من ميزات الصدفة، لكننا ركزّنا على الميزات التي تُستخدم مباشرةً في سطر الأوامر. تحتوي الصدفة أيضًا على مجموعة من الميزات التي تُستخدم عادةً (لكن ليس دائمًا) عند كتابة البرامج.

طريقة كتابة سكربت شِل

لكي نستطيع كتابة وتنفيذ سكربتات شِل بنجاح، نحتاج إلى أن نقوم بالأشياء الثلاثة الآتية:

- **كتابة السكربت.** سكربتات الشِل هي ملفات نصية عادية. لذا سنحتاج إلى محرر نصي لكتابتهم. أفضل المحررات هي تلك التي توافر تلون للأكواد، الذي يساعدنا في معرفة مكمن بعض الأخطاء الشائعة (نسيان إغلاق الأقواس...). محررات vim، و gedit، و kate وغيرها الكثير هي مثال جيد على المحررات المناسبة لكتابة السكربتات.
- **إعطاء إذن التنفيذ للسكربت.** النظام صارم جدًا (ولسبب منطقي) في عدم معاملة أي ملف نصي على أنه برنامج. يجب أن نعطي إذن التنفيذ للسكربت كي نستطيع تشغيله.
- **وضع ملف السكربت في مكانٍ معين كي تستطيع الصدفة العثور عليه.** تبحث الصدفة في

مجلدات معينة عن الملفات التنفيذية إذا لم يُوفَّر المسار الكامل لها. سنضع السكربتات في أحد تلك المجلدات.

صياغة السكربت

بالإبقاء على التقاليد البرمجية، سننشئ برنامج "أهلًا بالعالم" يشرح طريقة صياغة أبسط السكربتات. لنشغل محررنا المفضل ونكتب السكربت الآتي:

```
#!/bin/bash

# This is our first script.

echo 'Hello World!'
```

السطر الأخير من السكربت هو مألوفٌ لديك، مجرد الأمر echo مع سلسلة نصية تُمرَّر كوسيط. السطر الثاني هو مألوفٌ أيضًا، يبدو أنه تعليق كباقي التعليقات التي تظهر في العديد من ملفات الإعدادات التي تفحصناها أو عدّلناها. يجدر بالذكر أن التعليقات في سكربتات الشِل قد تظهر في نهاية الأسطر التي تحتوي على أوامر كالآتي:

```
echo 'Hello World!' # This is a comment too
```

ستتجاهل الصدفة كل شيء من الرمز # إلى نهاية السطر.

يمكن أيضًا استخدام التعليقات في سطر الأوامر (كالعديد من الأمور الأخرى):

```
[me@linuxbox ~]$ echo 'Hello World!' # This is a comment too
Hello World!
```

لكن ليس من الشائع استخدام التعليقات في سطر الأوامر، إلا أننا نستطيع ذلك.

أول سطر من السكربت غامض قليلًا. يبدو أنه تعليق لأنه يبدأ برمز #، لكن يبدو أنه له غاية أخرى. التعبير "!#" هو تعبير خاص يسمى shebang، يستخدم تعبير shebang لإعلام النظام باسم المفسر الذي يجب استخدامه لتنفيذ هذا الملف. يجب أن تحتوي جميع سكربتات الشِل على هذا التعبير كأول سطر فيها.

لنحفظ الآن ملف السكربت باسم `hello_world`.

أذونات التنفيذ

يجب علينا الآن أن نعطي أذونات التنفيذ للسكربت. يمكن أن يتم ذلك ببساطة باستخدام chmod:

```
[me@linuxbox ~]$ ls -l hello_world
-rw-r--r-- 1  me    me     63 2009-03-07 10:10 hello_world
[me@linuxbox ~]$ chmod 755 hello_world
[me@linuxbox ~]$ ls -l hello_world
-rwxr-xr-x 1  me    me     63 2009-03-07 10:10 hello_world
```

يوجد ضبطين شهيرين لإعدادات السكربتات؛ 755 للسكربتات التي يستطيع أي شخص في النظام تنفيذها، و 700 للسكربتات التي يستطيع تنفيذها المالك فقط. لاحظ أن السكربتات يجب أن تكون قابلة للقراءة لكي نستطيع تنفيذها.

مكان ملف السكربت

يمكننا الآن تنفيذ السكربت بعد أن حددنا الأذونات المناسبة له:

```
[me@linuxbox ~]$ ./hello_world
Hello World!
```

يجب أن نُسبِق اسم السكربت بمساره كي نستطيع تنفيذه. إذا لم نقم بذلك، فسوف نحصل على رسالة الخطأ الآتية:

```
[me@linuxbox ~]$ hello_world
bash: hello_world: command not found
```

لماذا حصل ذلك؟ ما هو الفرق ما بين هذا السكربت وباقي البرامج؟ لا يوجد أي اختلاف! المشكلة فقط في مكان تخزين السكربت. بالعودة إلى الفصل 11، ناقشنا متغير البيئة PATH وتأثيره على أماكن البحث عن البرامج التنفيذية. ملخص ذاك النقاش هو أن النظام يبحث في قائمة من المجلدات في كل مرة يحتاج فيها إلى البحث عن ملف تنفيذي إذا لم يُحَدَّد المسار الكامل للبرنامج. هذا هو السبب الذي يجعل النظام يعرف أن يُنفِّذ البرنامج /bin/ls عندما نُدخِل الأمر ls في سطر الأوامر. مجلد /bin هو أحد المجلدات التي يَبحث فيها النظام تلقائيًا. يُحتَفظ بقائمة المجلدات التي سيُبحَث فيها في متغير البيئة PATH مفصولةً بنقطتَين رأسيتَين، نستطيع أن نشاهد محتويات المتغير PATH:

```
[me@linuxbox ~]$ echo $PATH
```

```
/home/me/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin:/usr/games
```

إذا كان السكربت موجودًا في أحد تلك المجلدات، فستُحَلّ المشكلة. لاحظ أول مجلد من تلك المجلدات هو /home/me/bin. تضبط أغلب توزيعات لينُكس المتغير PATH لكي يحتوي على مجلد bin في مجلد المنزل الخاص بالمستخدمين للسماح لهم بتنفيذ البرامج الخاصة بهم. نستطيع إذًا تشغيل السكربت كأي برنامج آخر إذا أنشأنا مجلد bin ووضعنا السكربت فيه:

```
[me@linuxbox ~]$ mkdir bin
[me@linuxbox ~]$ mv hello_world bin
[me@linuxbox ~]$ hello_world
Hello World!
```

إذا لم يكن يحتوي متغير PATH على ذاك المجلد، فنستطيع إضافته بسهولة بإضافة السطر الآتي إلى ملف ".bashrc":

```
export PATH=~/bin:"$PATH"
```

سيأخذ هذا التغيير مفعوله بعد بدء جلسة طرفية جديدة. سنجعل الصدفة تعيد قراءة الملف bashrc. لتطبيق الإعدادات الجديدة دون إنهاء الجلسة:

```
[me@linuxbox ~]$ . .bashrc
```

رمز النقطة هو اختصار للأمر source، الذي هو أمر مدمج بالصدفة يقرأ ملف يحتوي على الأوامر ويعاملها كأنها أُدخلت من لوحة المفاتيح.

ملاحظة: يضيف أوبنتو المجلد bin/~ إلى متغير PATH إذا كان المجلد bin/~ موجودًا عند تنفيذ ملف ".bashrc".

أماكن جيدة لحفظ السكربتات

المجلد bin/~ هو مكان جيد لوضع السكربتات التي كُتِبَت للاستخدام الشخصي. إذا كتبنا سكربتًا يُسمَح لجميع مستخدمي النظام بتنفيذه، فسنضعه في مجلد /usr/local/bin. توضع عادةً السكربتات التي كُتِبَت للاستخدام من قِبل مدير النظام في مجلد /usr/local/sbin. يجب أن توضع البرمجيات التي أُنشئت محليًا (سواءً كانت سكربتات أو برامج مبنية من المصدر) في مجلد /usr/local وليس في bin/أو

‎/usr/bin‎. يجب أن تحتوي تلك المجلدات على الملفات التي يتم تزويدها من قِبل صانعي التوزيعة وذلك وفق معايير شجرة نظام الملفات في لينُكس.

بعض حيل تنسيق الأكواد

أحد الشروط المهمة التي يجب أن يستوفيها السكربت هو سهولة صيانته؛ أي سهولة تعديل السكربتات من قِبل كاتبها أو الآخرين لكي يتلاءم مع التغيرات الضرورية. جعل السكربت سهل القراءة والفهم هو إحدى الطرق التي تحقق سهولة الصيانة.

استخدام الخيارات الطويلة

توفر أغلب الأوامر التي درسناها خياراتٍ طويلةً وقصيرةً. على سبيل المثال، يحتوي الأمر ls على العديد من الخيارات التي يمكن التعبير عنها بالشكل القصير أو الطويل. مثلًا، الأمر:

```
[me@linuxbox ~]$ ls -ad
```

والأمر:

```
[me@linuxbox ~]$ ls --all --directory
```

هما أمران متساويان. ولغرض تقليل الكتابة، تكون الخيارات القصيرة هي المُفضَّلة عند استخدام الخيارات في سطر الأوامر، لكن عند كتابة السكربتات، تزيد الخيارات الطويلة من سهولة القراءة.

المحاذاة والإكمال السطري

يمكن تقسيم الأمر إلى عدّة أسطر عند كتابة خيارات الأوامر الطويلة لزيادة قابلية قراءة الملف. ألقينا نظرة على مثال طويل عن الأمر find في الفصل السابع عشر:

```
[me@linuxbox ~]$ find playground \( -type f -not -perm 0600 -exec
chmod 0600 ‘{}’ ‘;’ \) -or \( -type d -not -perm 0711 -exec chmod
0711 ‘{}’ ‘;’ \)
```

كما هو واضح، من الصعب فهم الأمر السابق من الوهلة الأولى. أما عند كتابة الأمر السابق في سكربت، فيصبح من السهل فهم آلية عمل ذاك الأمر عند كتابته بهذه الطريقة:

```
find playground \
```

```
        \( \
                -type f \
                -not -perm 0600 \
                -exec chmod 0600 ‘{}’ ‘;’ \
        \) \
        -or \
        \( \
                -type d \
                -not -perm 0711 \
                -exec chmod 0711 ‘{}’ ‘;’ \
        \)
```

باستخدام مكملات الأسطر (بالشرطة المائلة الخلفية في آخر السطر) والمحاذاة، أصبحت الآلية التي يستخدمها الأمر السابق سهلة الفهم بالنسبة إلى القارئ. هذه التقنية تعمل في سطر الأوامر أيضًا لكن قل ما تُستخدم، لأنها صعبة الكتابة والتعديل. أحد الفروق ما بين السكربتات وسطر الأوامر هو إمكانية استخدام مسافات الجدولة لمحاذاة النص في السكربتات، لكن لا يمكن فعل ذلك في سطر الأوامر، لأن زر tab في سطر الأوامر يُستخدَم للإكمال التلقائي.

إعداد vim لكتابة السكربتات

لدى المحرر vim الكثير من إعدادات الضبط. توجد عدّة خيارات شائعة تُستخدم عند كتابة السكربتات:

:syntax on

يُفعّل الأمر السابق تلوين الأكواد. يظهر كل عنصر من العناصر المكونة للسكربت بلون مختلف عند عرض سكربتٍ ما مع استخدام هذا الضبط. الذي يسهّل التعرف على بعض أنواع الأخطاء البرمجية. لاحظ أنك تحتاج إلى النسخة الكاملة من vim لاستخدام هذا الخيار، بالإضافة إلى وجود "!#" في بداية الملف لكي يعرف vim أن الملف الذي يُعدَّل هو سكربت شِل. إذا واجهت بعض الصعوبات في الأمر السابق، فجرب **:set syntax=sh** عوضًا عنه.

:set hlsearch

تفعيل خيار تعليم (أو تحديد) نتائج البحث. لنفترض أننا بحثنا عن الكلمة "echo" بعد تفعيل هذا الخيار، فسنجد أن كل كلمة "echo" في الملف قد عُلِّمَت.

:set tabstop=4

تحديد عدد الحقول التي ستحتلها مسافة الجدولة. الخيار الافتراضي هو ستة حقول. ضبط هذه القيمة إلى أربعة (وهو أمرٌ شائعٌ بين المطورين) يسمح بتوسيع الأسطر الطويلة داخل الشاشة بشكلٍ

أسهل.

```
:set autoindent
```

تفعيل المحاذاة التلقائية. أي سيقوم vim بمحاذاة السطر الجديد بنفس محاذاة السطر الحالي. وهذا ما يُسرِّع كتابة مختلف البُنى في أغلب لغات البرمجة. لإيقاف المحاذاة التلقائية، اضغط على Ctrl-d.

يمكنك أن تجعل هذه الإعدادات افتراضيةً بإضافتها إلى ملف vimrc.~ لكن دون استخدام النقطتَين الرأسيتَين قبلها.

الخلاصة

في أولِ فصلٍ لنا عن كتابة السكربتات، تعرفنا على طريقة كتابة السكربتات وتنفيذها بسهولة على النظام. وتعرفنا أيضًا على العديد من تقنيات التنسيق لزيادة قابلية قراءة (وبالتالي صيانة) السكربتات.

الفصل الخامس والعشرون:

بدء المشروع

سنبني بدءًا من هذا الفصل برنامجًا خاصًا بنا. غَرض هذا المشروع هو تعلم طريقة استخدام مختلف ميزات الصدفة لإنشاء البرامج، بالإضافة إلى طريقة كتابة برامج "جيدة".

البرنامج الذي سنكتبه هو مولد تقارير (report generator)؛ يظهر مختلف الإحصائيات عن النظام، وحالته. ويحفظ الناتج بصيغة HTML، التي يمكن عرضها باستخدام متصفح وِب كفَيرفُكس أو كروم.

تبنى البرامج عادةً بسلسلة من الخطوات، حيث تضاف الميزات وتُحسَّن إمكانيات البرنامج في كل مرحلة. أول مرحلة من برنامجنا هي إنشاء مستند HTML مُصغّر لا يحتوي على أية معلومات عن النظام بعد.

المرحلة الأولى: المستند المُصغّر

يجب علينا أولًا أن نتعرف على الشكل المُنظَّم لمستندات HTML. تكون بنية ملف HTML الأساسية كالآتي:

```
<HTML>
	<HEAD>
		<TITLE>Page Title</TITLE>
	</HEAD>
	<BODY>
		Page body.
	</BODY>
</HTML>
```

إذا أدخلنا هذا النص في المحرر النصي المفضل لدينا وحفظناه باسم `foo.html`، فإننا نستطيع استخدام الرابط الآتي في فَيرفُكس لعرضه:

```
file:///home/username/foo.html
```

أول مرحلة من مراحل بناء برنامجنا هي جعله قادرًا على إخراج ملف HTML السابق إلى مجرى الخرج القياسي. يمكننا بكل سهولة كتابة هذا البرنامج. لنقم الآن بإنشاء وتحرير ملف باسم `~/bin/sys_info_page`:

```
[me@linuxbox ~]$ vim ~/bin/sys_info_page
```

أدخل الآن البرنامج الآتي:

```
#!/bin/bash

# Program to output a system information page

echo   "<HTML>"
echo   "       <HEAD>"
echo   "               <TITLE>Page Title</TITLE>"
echo   "       </HEAD>"
echo   "       <BODY>"
echo   "               Page body."
echo   "       </BODY>"
echo   "</HTML>"
```

أولى محاولاتنا لحل هذه المشكلة تتضمن استخدام shebang ("!#")، وتعليق (دائمًا تكون إضافة التعليقات هي فكرة جيدة)، ومجموعة من أوامر echo، كل أمر يطبع سطرًا واحدًا فقط. لنمنح إذن التنفيذ للسكربت ونجرّب تشغيله بعد حفظه:

```
[me@linuxbox ~]$ chmod 755 ~/bin/sys_info_page
[me@linuxbox ~]$ sys_info_page
```

بعد تنفيذ البرنامج، ستظهر محتويات مستند HTML على الشاشة، لأن أوامر echo الموجودة في السكربت تُرسِل مخرجاتها إلى مجرى الخرج القياسي. سنُعيد تنفيذ البرنامج لكن هذه المرّة سنُعيد توجيه المخرجات إلى ملف sys_info_page.html لكي نستطيع معاينة الناتج في متصفح الوِب:

```
[me@linuxbox ~]$ sys_info_page > sys_info_page.html
[me@linuxbox ~]$ firefox sys_info_page.html
```

كل شيء على ما يرام حتى الآن.

من الجيد أن أن نطمح دائمًا إلى السهولة والوضوح عند كتابة برامجنا. صيانة البرامج تكون أسهل بكثير عندما تسهل قراءة وفهم السكربت. نسختنا الحالية من البرنامج تعمل جيدًا، لكن يمكن أن تُكتَب بشكل أبسط. يمكننا دمج كل أوامر echo بأمر واحد فقط، مما يسهل إضافة أسطر جديدة إلى البرنامج. لنعدل برنامجنا ليصبح كالآتي:

```
#!/bin/bash
```

```
# Program to output a system information page

echo "<HTML>
        <HEAD>
               <TITLE>Page Title</TITLE>
        </HEAD>
        <BODY>
               Page body.
        </BODY>
</HTML>"
```

يمكن أن تحتوي أيّة سلسلة نصية على محارف الانتقال إلى سطرٍ جديد، مما يساعد في إخراج نص متعدد الأسطر. ستستمر الصدفة في قراءة النص حتى يُغلَق الاقتباس. يمكن استخدام هذه الحيلة في سطر الأوامر أيضًا:

```
[me@linuxbox ~]$ echo "<HTML>
>            <HEAD>
>                      <TITLE>Page Title</TITLE>
>            </HEAD>
>            <BODY>
>                      Page body.
>            </BODY>
> </HTML>"
```

الرمز ">" الذي يظهر في بداية السطر هو قيمة متغير الصدفة PS2. ويظهر عندما نكتب تعبير متعدد الأسطر في الصدفة. قد تكون هذه الميزة غامضة بعض الشيء، لكنك ستجد أنها مفيدة للغاية بعد أن نشرح كيفية كتابة تعابير برمجية متعددة الأسطر.

المرحلة الثانية: إضافة بعض البيانات

لنضع في برنامجنا بعض البيانات بعد أن أصبح يظهر مستند HTML مصغر. سنطبق الآن التغيرات الآتية:

```
#!/bin/bash

# Program to output a system information page
```

```
echo "<HTML>
          <HEAD>
                    <TITLE>System Information Report</TITLE>
          </HEAD>
          <BODY>
                    <H1>System Information Report</H1>
          </BODY>
</HTML>"
```

أضفنا عنوانًا (TITLE) وترويسةً (H1) للتقرير.

المتغيرات والثوابت

هنالك إشكالية صغيرة في السكربت الخاص بنا. لاحظ كيف تكررت السلسلة النصية "System Information Report" أكثر من مرّة؟ هذه ليست بالمشكلة في هذا السكربت، لكن ماذا لو كان السكربت طويلًا وتكررت هذه العبارة أكثر من مرّة في أكثر من موضع. إذا أردنا أن نغير العنوان إلى عبارة أخرى فسنحتاج إلى تغييرها في عدّة مواضع، مما قد يتطلب عملًا كثيرًا. لكن ماذا لو عدّلنا في السكربت حتى تصبح تلك العبارة موجودةً في مكان واحد فقط؟ هذا ما يجعل الصيانة المستقبلية للسكربت أسهل وأسرع:

```
#!/bin/bash

# Program to output a system information page

title="System Information Report"

echo "<HTML>
           <HEAD>
                    <TITLE>$title</TITLE>
           </HEAD>
           <BODY>
                    <H1>$title</H1>
           </BODY>
</HTML>"
```

بإنشائنا للمتغير (variable) ذي الاسم title وإسناد القيمة "System Information Report" إليه،

استطعنا الاستفادة من توسعة المتغيرات لكي نجعل العنوان يظهر في عدّة مواضع دون تكرار عبارة العنوان في كل مرّة.

إذًا، كيف ننشئ متغيرًا؟ بكل بساطة، نستخدمه! عندما تواجه الصدفة متغيرًا جديدًا، فستُنشئه تلقائيًا. وهذا ما يختلف عن العديد من لغات البرمجة في أن المتغيرات يجب أن يُعلَن عنها (declared) أو تُعرَّف قبل استخدامها. الصدفة متساهلة في هذا الموضوع، لكنه قد يؤدي إلى بعض المشاكل. على سبيل المثال، افترض أننا جرّبنا السيناريو الآتي في سطر الأوامر:

```
[me@linuxbox ~]$ foo="yes"
[me@linuxbox ~]$ echo $foo
yes
[me@linuxbox ~]$ echo $fool
[me@linuxbox ~]$
```

أسندنا في بادئ الأمر القيمة "yes" إلى المتغير foo، ومن ثم عرضنا قيمته باستخدام echo. جرّبنا بعد ذلك عرض قيمة المتغير (الذي ارتكبنا خطأً إملائيًا في اسمه) fool ولكننا حصلنا على نتيجة فارغة؛ حصل ذلك لأن الصدفة أنشأت، وبكل سرور، متغيرًا جديدًا باسم fool وأسندت إليه قيمةً فارغةً. من المهم أيضًا فهم ماذا حصل فعلًا في المثال السابق. من خبرتنا السابقة مع آلية توسيع المعاملات التي تقوم بها الصدفة، نعلم أن الأمر:

```
[me@linuxbox ~]$ echo $foo
```

سيؤدي إلى توسعة المتغير foo ويعطي الناتج:

```
[me@linuxbox ~]$ echo yes
```

بينما الأمر:

```
[me@linuxbox ~]$ echo $fool
```

سيُوسَّع إلى:

```
[me@linuxbox ~]$ echo
```

وُسِّع المتغير ذو القيمة الفارغة إلى "لا شيء"! يمكننا أن نعبث قليلًا مع الأوامر التي تتطلب وسائط. كالمثال الآتي:

```
[me@linuxbox ~]$ foo=foo.txt
[me@linuxbox ~]$ foo1=foo1.txt
[me@linuxbox ~]$ cp $foo $fool
cp: missing destination file operand after `foo.txt'
Try `cp --help' for more information.
```

أسندنا قيمتَين إلى المتغيرَين foo و foo1. ونفّذنا بعدها الأمر cp، لكن أخطأنا في تهجئة الوسيط الثاني. سيحصل الأمر cp على وسيط واحد بعد التوسعة بينما يتطلب وجود وسيطين.

هنالك بعض القواعد حول أسماء المتغيرات:

- يمكن أن تحتوي أسماء المتغيرات على أحرف وأرقام وعلى الشرطة السفلية.
- يجب أن يكون أول محرف في اسم المتغير حرفًا أبجديًا أو شرطةً سفليةً.
- لا يُسمَح بوجود الفراغات وعلامات الترقيم في أسماء المتغيرات.

كلمة "المتغير" تعني القيمة التي تتغير، وتستخدم المتغيرات في أغلب البرامج بهذه الطريقة. لكن أُستخدم المتغير title في برنامجنا كثابت (constant). الثابت -كما المتغير- يتألف من اسم وقيمة. لكن الفرق بينهما هو أن قيمة الثابت لا تتغير. في تطبيقٍ يقوم بالعمليات الحسابية، ربما نُعرِّف PI كثابت، ونسند القيمة 3.1415 إليه؛ عوضًا عن استخدام القيمة مباشرةً (مما يزيد وضوح برنامجنا). لا تُفرِّق الصدفة بين المتغيرات والثوابت؛ لكن التفرقة بينهما تلائم المبرمجين؛ إحدى القواعد هي تسمية جميع الثوابت بأحرف كبيرة، وجميع المتغيرات بأحرف صغيرة. باستطاعتنا تعديل السكربت لملائمة هذه القواعد:

```
#!/bin/bash

# Program to output a system information page

TITLE="System Information Report For $HOSTNAME"
echo "<HTML>
		<HEAD>
				<TITLE>$TITLE</TITLE>
		</HEAD>
		<BODY>
				<H1>$TITLE</H1>
		</BODY>
</HTML>"
```

لقد استثمرنا الفرصة كي نُحسّن العنوان بإضافة قيمة متغير الصدفة HOSTNAME الذي يمثل اسم الحاسوب على الشبكة.

ملاحظة: في الواقع، توفر الصدفة طريقةً لإنشاء ثوابت باستخدام الأمر المضمن `declare` مع الخيار `-r` (read-only). بإمكاننا إسناد قيمة إلى TITLE كالآتي:

```
declare -r TITLE="Page Title"
```

ستمنع الصدفة أيّة تعديلات على الثابت TITLE. تُستخدم هذه الميزة نادرًا. لكنها موجودة لاستخدامها مع بعض السكربتات الرسمية جدًا.

إسناد القيم إلى المتغيرات والثوابت

بدأت معرفتنا بالتوسعات تؤتي أُكلها الآن. كما شاهدنا سابقًا، تُسنَد القيم إلى المتغيرات كالآتي:

```
variable=value
```

حيث `variable` هو اسم المتغير و `value` هي سلسلة نصيّة. وعلى النقيض من لغات البرمجة الأخرى. لا تلقي الصدفة بالًا لنوع البيانات المُسنَدة إلى المتغيرات؛ ستعاملهم جميعًا على أنهم سلاسل نصيّة. يمكن أن نجبر الصدفة على جعل القيمة المسندة إلى المتغيرات محصورةً على الأعداد الصحيحة باستخدام الأمر `declare` مع الخيار `-i`. لكن هذه الميزة نادرة الاستخدام (كجعل المتغيرات للقراءة فقط).

لاحظ عدم وجود فراغات بين اسم المتغير، وإشارة المساواة، والقيمة المسندة في تعبير الإسناد.

ما الذي يمكن أن تحويه القيمة؟ أيّ شيء قابل للتوسع إلى سلسلة نصيّة:

```
a=z                   # Assign the string "z" to variable a.
b="a string"          # Embedded spaces must be within quotes.
c="a string and $b"   # Other expansions such as variables can be
                      # expanded into the assignment.
d=$(ls -l foo.txt)    # Results of a command.
e=$((5 * 7))          # Arithmetic expansion.
f="\t\ta string\n"    # Escape sequences such as tabs and newlines.
```

يمكن إجراء أكثر من عملية إسناد في سطرٍ واحد.

```
a=5 b="a string"
```

أثناء التوسعة، يمكن أن تحاط أسماء المتغيرات اختياريًا بالقوسَين المعقوفَين "{}". يمكننا الاستفادة منهما

عندما يصبح اسم المتغير ملتبسًا بسبب ما يحيط به. حاولنا في المثال الآتي أن نُغيّر اسم الملف من `myfile` إلى `myfile1`:

```
[me@linuxbox ~]$ filename="myfile"
[me@linuxbox ~]$ touch $filename
[me@linuxbox ~]$ mv $filename $filename1
mv: missing destination file operand after `myfile'
Try `mv --help' for more information.
```

فشلت هذه المحاولة لأن الصدفة تُفسِّر الوسيط الثاني للأمر mv على أنه متغير جديد (وفارغ). يمكن حلّ هذه المشكلة بالطريقة الآتية:

```
[me@linuxbox ~]$ mv $filename ${filename}1
```

لم تعتبر الصدفة الرقم 1 أنه جزء من اسم المتغير بعد إضافتنا للقوسَين المعقوفَين.

سنستثمر هذه الفرصة لإضافة بعض البيانات إلى تقريرنا. تحديدًا، الوقت والتاريخ الذي أُنشئ هذا التقرير عنده، واسم المستخدم:

```
#!/bin/bash

# Program to output a system information page

TITLE="System Information Report For $HOSTNAME"
CURRENT_TIME=$(date +"%x %r %Z")
TIMESTAMP="Generated $CURRENT_TIME, by $USER"

echo "<HTML>
          <HEAD>
                    <TITLE>$TITLE</TITLE>
          </HEAD>
          <BODY>
                    <H1>$TITLE</H1>
                    <P>$TIMESTAMP</P>
          </BODY>
</HTML>"
```

Here Documents

شرحنا طريقتَيـن مختلفتَيـن لإخـراج النص، كلاهما باستخدام الأمـر echo. هنالك طريقة أُخـرى تسمى here Document أو here script. التي هـي شـكل إضـافي لإعـادة تـوجيه الـدخل أو الخـرج التـي نُضمِّن فيهـا النص داخل السكربت ثم نمرره لمجرى الدخل القياسي للأمر. تعمل كالآتي:

```
command << token

text

token
```

حيث command هو اسم الأمر الذي يقبل المدخلات من مجرى الدخل القياسي. و token هي سلسلة نصيّة تُستخدم للإشارة إلى نهاية النص المُضمَّن. سنُعدِّل السكربت كي يستخدم here document:

```
#!/bin/bash

# Program to output a system information page

TITLE="System Information Report For $HOSTNAME"
CURRENT_TIME=$(date +"%x %r %Z")
TIMESTAMP="Generated $CURRENT_TIME, by $USER"

cat << _EOF_
<HTML>
          <HEAD>
                    <TITLE>$TITLE</TITLE>
          </HEAD>
          <BODY>
                    <H1>$TITLE</H1>
                    <P>$TIMESTAMP</P>
          </BODY>
</HTML>
_EOF_
```

استخدام السكربت السابق الأمر cat و here document بدلًا من الأمر echo. قد اختيرت السلسلة النصيّة _EOF_ (الـتي ترمـز إلـى "End Of File" أي نهايـة الملـف. وهـي اختصـار شـائع) علـى أنهـا "العلامـة الرمزيـة" (token)، التي تُشير إلى نهاية النص المُضمَّن. لاحظ أن العلامة الرمزية يجب أن تظهر مُنفَرِدَةً في السطر دون أن تتبعها أيّة فراغات.

إذًا، ما هي ميزات استخدام here document؟ هي تقريبًا كالأمر echo إلا أن علامات الاقتباس المفردة والمزدوجة تفقد معناها الخاص افتراضيًّا. هذا مثالٌ في سطر الأوامر:

```
[me@linuxbox ~]$ foo="some text"
[me@linuxbox ~]$ cat << _EOF_
> $foo
> "$foo"
> '$foo'
> \$foo
> _EOF_
some text
"some text"
'some text'
$foo
```

كما لاحظنا، لا تلقي الصدفة بالًا لعلامات الاقتباس. فهي تعاملها كأي محرف عادي وهذا ما يمكّننا من تضمين علامات الاقتباس "بحريّة" داخل here document. وهذا ما يفيدنا للغاية في برنامج التقارير الخاص بنا.

يمكن استخدام here document مع أيّ أمر يقبل المدخلات من مجرى الدخل القياسي. سنستخدم here document في المثال الآتي لتمرير سلسلة من الأوامر إلى برنامج ftp كي نُنَزِّلَ ملفًا من خادم ftp بعيد:

```
#!/bin/bash

# Script to retrieve a file via FTP

FTP_SERVER=ftp.nl.debian.org
FTP_PATH=/debian/dists/lenny/main/installer-i386/current/images/cdrom
REMOTE_FILE=debian-cd_info.tar.gz

ftp -n << _EOF_
open $FTP_SERVER
user anonymous me@linuxbox
cd $FTP_PATH
hash
get $REMOTE_FILE
bye
_EOF_
```

```
ls -l $REMOTE_FILE
```

إذا غيّرنا معامل إعادة التوجيه من "<<" إلى "-<<" فستتجاهل الصدفة مسافات الجدولة البادئة في here document. وهذا ما يسمح لنا بمحاذاة النص مما يزيد من قابلية قراءته:

```
#!/bin/bash

# Script to retrieve a file via FTP

FTP_SERVER=ftp.nl.debian.org
FTP_PATH=/debian/dists/lenny/main/installer-i386/current/images/cdrom
REMOTE_FILE=debian-cd_info.tar.gz

ftp -n <<- _EOF_
	open $FTP_SERVER
	user anonymous me@linuxbox
	cd $FTP_PATH
	hash
	get $REMOTE_FILE
	bye
	_EOF_
ls -l $REMOTE_FILE
```

الخلاصة

بدأنا في هذا الفصل مشروعًا سيحفزنا أثناء مرحلة بناء سكربت ناجح. لقد تعرفنا على مفهوم المتغيرات والثوابت وكيف يمكن توظيفهما في السكربتات. هما تطبيق من التطبيقات العديدة لتوسعة المتغيرات. ألقينا نظرةً أيضًا على آلية توليد مخرجات من السكربت، وتضمين مقاطع نصيّة.

الفصل السادس والعشرون:

نمط التصميم Top-Down

من المؤكد أن صعوبة تصميم البرامج وكتابتها وصيانتها ستزداد كلما كبرت وازدادت تعقيدًا. غالبًا ما تُجزَّأ المهام الكبيرة والمعقدة في المشاريع الكبيرة إلى مهام أصغر وأبسط. لنفترض أننا نريد وصف مهمة يومية شائعة، كالذهاب إلى السوق وشراء الطعام، إلى "شخص" من كوكب المريخ. يمكننا شرح العملية كاملةً له بسلسلة من الخطوات:

1. اركب في السيارة.
2. قم بقيادة السيارة إلى السوق.
3. اركن السيارة.
4. ادخل إلى السوق.
5. اشترِ الطعام.
6. قم بالعودة إلى السيارة.
7. قم بالقيادة إلى المنزل.
8. اركن السيارة.
9. ادخل إلى المنزل.

لكن سيحتاج ذاك الشخص من المريخ إلى المزيد من التفاصيل. يمكننا تجزئة المهمة "اركن السيارة" إلى سلسلة من الخطوات:

1. أوجد مساحة فارغة لركن السيارة.
2. قم بقيادة السيارة إلى تلك المساحة.
3. أطفئ المحرك.
4. ارفع المكابح اليدوية.
5. اخرج من السيارة.
6. اقفل السيارة.

المهمة الفرعية "أطفئ المحرك" يمكن تجزئتها إلى عدّة خطوات تتضمن "أطفئ دارة الإشعال"، و "أخرج

مفتاح السيارة" وهكذا؛ حتى تُفصَّل كل خطوة من كامل عملية الذهاب إلى السوق.

عملية تعريف الخطوط الأساسية ومن ثم كتابة التفاصيل لتلك الخطوات تسمى "نمط تصميم Top-Down". يسمح هذا النمط لنا بتقسيم المهام الكبيرة والضخمة إلى مهام أبسط وأصغر. نمط Top-Down هو طريقة شائعة في تصميم البرامج وهو متلائم كثيرًا مع برمجة الشِل.

سنستخدم في هذا الفصل نمط التصميم Top-Down لكي نكمل تطوير سكربت توليد التقارير.

دوال الشِل

يولّد السكربت حاليًا مستند HTML باتباع هذه الخطوات:

1. ابدأ مستند HTML (وسم البداية <html>).
2. افتح رأس الصفحة (وسم البداية <head>).
3. اضبط عنوان المستند (الوسم <title>).
4. أغلق رأس الصفحة (وسم الإغلاق </head>).
5. افتح جسد الصفحة (وسم البداية <body>).
6. افتح ترويسة الصفحة (الوسم <h1>).
7. اطبع بصمة الوقت.
8. أغلق جسد الصفحة (وسم الإغلاق </body>).
9. أغلق مستند HTML (وسم الإغلاق </html>).

سنضيف مهامًا أُخرى بين الخطوتين 7 و 8 في مرحلة التطوير القادمة، التي تتضمن:

- الوقت الذي مضى على تشغيل النظام والحِمل المُطبَّق عليه (أي متوسط عدد المهام التي تعمل على المعالج، في فواصل زمنية محددة).
- مساحة القرص. أي المساحة التخزينية المستخدمة من قِبل أقراص التخزين.
- مساحة المنزل. أي المساحة التخزينية المستخدمة من قِبل المستخدمين.

إذا كان لدينا أمر جاهز لكل مهمة من تلك المهام، فنستطيع إضافته مباشرةً إلى السكربت:

```
#!/bin/bash

# Program to output a system information page
```

```
TITLE="System Information Report For $HOSTNAME"
CURRENT_TIME=$(date +"%x %r %Z")
TIME_STAMP="Generated $CURRENT_TIME, by $USER"

cat << _EOF_
<HTML>
          <HEAD>
                    <TITLE>$TITLE</TITLE>
          </HEAD>
          <BODY>
                    <H1>$TITLE</H1>
                    <P>$TIME_STAMP</P>
                    $(report_uptime)
                    $(report_disk_space)
                    $(report_home_space)
          </BODY>
</HTML>
_EOF_
```

باستطاعتنا إنشاء الأوامر الإضافية بطريقتَين. يمكننا كتابة ثلاثة سكربتات منفصلة ووضعها في مجلد يحتوي المتغير PATH على مساره؛ وبإمكاننا تضمين برنامجنا على شكل دوال شِل. وكما ذكرنا سابقًا، دوال الشِل هي سكربتات مصغرة موجودة داخل سكربتات أخرى وتعمل كبرامج مستقلة. لدى دوال الشِل الشكلَين الأساسيَين الآتيَين:

```
function name {
        commands
        return
}
```

و:

```
name () {
        commands
        return
}
```

حيث name هو اسم الدالة و commands هو سلسلة الأوامر المحتواة داخل الدالة. كلا الشكلَين متساوي ويمكن التبديل بينهما دون مشاكل. يوضح السكربت الآتي استخدام دوال الشِل:

```
1     #!/bin/bash
2
3     # Shell function demo
4
5     function funct {
6             echo "Step 2"
7             return
8     }
9
10     # Main program starts here
11
12     echo "Step 1"
13     funct
14     echo "Step 3"
```

عندما تقرأ الصدفة السكربت، فستتغاضى عن الأسطر من 1 إلى 11، حيث تحتوي هذه الأسطر على تعليقات وتعريف الدالة. يبدأ التنفيذ من السطر 12 الذي يحتوي على الأمر echo. السطر 13 يستدعي الدالة funct وتُنفِّذ الصدفة الدالة كأي أمرٍ آخر؛ حيث ينتقل تنفيذ البرنامج إلى السطر 6، وسيُنفَّذ الأمر echo. وبعدها سينتقل إلى السطر 7 حيث سيُنهي الأمر return الدالة، وسيعود التنفيذ إلى السطر 14؛ حيث ينفذ آخر أمر الذي هو echo. لاحظ أن تعريفات الدوال يجب أن تكون في أول السكربت قبل استدعائها.

سنضيف الآن تعريفات مُصغَّرة لدوال الشِل إلى سكربتنا:

```
#!/bin/bash

# Program to output a system information page

TITLE="System Information Report For $HOSTNAME"
CURRENT_TIME=$(date +"%x %r %Z")
TIME_STAMP="Generated $CURRENT_TIME, by $USER"

report_uptime () {
        return
```

```
}
report_disk_space () {
        return
}
report_home_space () {
        return
}

cat << _EOF_
<HTML>
        <HEAD>
                <TITLE>$TITLE</TITLE>
        </HEAD>
        <BODY>
                <H1>$TITLE</H1>
                <P>$TIME_STAMP</P>
                $(report_uptime)
                $(report_disk_space)
                $(report_home_space)
        </BODY>
</HTML>
_EOF_
```

قواعد تسمية الدوال هي نفسها قواعد تسمية المتغيرات. يجدر بالذكر أن الدالة يجب أن تحتوي أمرًا واحدًا على الأقل. الأمر return يفي بهذا الغرض (مع أنه اختياري).

المتغيرات المحليّة

في جميع السكربتات التي كتبناها إلى الآن، كانت جميع المتغيرات (بما فيها الثوابت) متغيراتٍ عامة. تحافظ المتغيرات العامة على وجودها في كامل البرنامج. هذا الأمر جيد في معظم الأحيان، لكن ذلك قد يُعقِّد استخدام دوال الشِل. من المفيد إنشاء متغيرات محليّة داخل الدوال. لا يمكن الوصول إلى المتغيرات المحلية إلا من داخل الدالة المُعرَّفة داخلها، وينعدم وجودها عند انتهاء الدالة.

يسمح وجود المتغيرات المحلية للمبرمج بأن يستخدم أسماء متغيرات موجودة مسبقًا سواءً داخل السكربت الأصلي أو داخل الدوال الأخرى، دون القلق من التضارب في الأسماء.

هذا مثال يشرح تعريف واستخدام المتغيرات المحلية:

```
#!/bin/bash

# local-vars: script to demonstrate local variables

foo=0                 # global variable foo

funct_1 () {

        local foo      # variable foo local to funct_1

        foo=1
        echo "funct_1: foo = $foo"
}

funct_2 () {

        local foo      # variable foo local to funct_2

        foo=2
        echo "funct_2: foo = $foo"

}

echo "global:  foo = $foo"
funct_1
echo "global:  foo = $foo"
funct_2
echo "global:  foo = $foo"
```

كما لاحظنا، تُعرَّف المتغيرات المحلية بإسباق اسمها بالكلمة "`local`"، وهذا ما يجعل المتغير محليًّا في الدالة التي أُنشئ فيها. أي أن هذا المتغير لن يكون مُعرَّفًا خارج الدالة. سنحصل على النتائج الآتية عندما نُنفِّذ السكربت السابق:

```
[me@linuxbox ~]$ local-vars
```

```
global:  foo = 0
funct_1: foo = 1
global:  foo = 0
funct_2: foo = 2
global:  foo = 0
```

نلاحظ أنَّ إسناد القيم إلى المتغير foo داخل دوال الشِل لا يؤثر أبدًا على قيمة المتغير foo المُعرَّف خارجها.

تسمح هذه الميزة بكتابة دوال شِل مستقلة عن بعضها البعض. توجد أيضًا خاصية مهمة لها هي منع أي جزء من البرنامج من التأثير على الجزء الآخر، وهذا ما يسمح بكتابة دوال شِل محمولة، أي يمكن نسخها ولصقها في سكربت آخر دون مشاكل.

إبقاء السكربتات قابلة للتشغيل

من المفيد في أثناء تطويرنا للبرنامج أن نبقيه في حالة قابلة للتشغيل. فعند إجراء تجارب متكررة عليه، نستطيع أن نعرف أخطاءه في أقرب وقتٍ ممكن، وهذا ما يجعل عملية تنقيح (debugging) الكود أسهل بكثير. على سبيل المثال، إذا نُفِّذَ البرنامج بنجاح ثم أجرينا تعديلًا صغيرًا عليه، وثم نفذنا البرنامج مرّةً أُخرى وظهرت مشكلة ما، فأغلب الظن أن التعديل الأخير هو سبب تلك المشكلة. بإضافتنا للدوال الفارغة، استطعنا التحقق من البنية المنطقية للبرنامج في مرحلة مبكرة. من الجيد أيضًا أن نجعل تلك الدوال تطبع عبارات معينة كي يعرف المبرمج الأعمال التي يقوم بها السكربت. إذا ألقينا نظرةً على المخرجات الحالية للسكربت:

```
[me@linuxbox ~]$ sys_info_page
<HTML>
        <HEAD>
               <TITLE>System Information Report For twin2</TITLE>
        </HEAD>
        <BODY>
               <H1>System Information Report For linuxbox</H1>
               <P>Generated 03/19/2009 04:02:10 PM EDT, by me</P>




        </BODY>
</HTML>
```

فسنلاحظ وجود بعض الأسطر الفارغة بعد بصمة الوقت، لكننا لسنا متأكدين من السبب. إذا عدّلنا الدوال لكي

تُظهِر بعض المخرجات:

```
report_uptime () {
          echo "Function report_uptime executed."
          return
}

report_disk_space () {
          echo "Function report_disk_space executed."
          return
}

report_home_space () {
          echo "Function report_home_space executed."
          return
}
```

وجرّبنا السكربت مرّة أُخرى:

```
[me@linuxbox ~]$ sys_info_page
<HTML>
         <HEAD>
                <TITLE>System Information Report For linuxbox</TITLE>
         </HEAD>
         <BODY>
                <H1>System Information Report For linuxbox</H1>
                <P>Generated 03/20/2009 05:17:26 AM EDT, by me</P>
                Function report_uptime executed.
                Function report_disk_space executed.
                Function report_home_space executed.
         </BODY>
</HTML>
```

فسنلاحظ أن الدوال الثلاث قد نُفِّذت.

بعد التحقق من عمل الدوال؛ حان الوقت لكي نكتب الأكواد التي تتحكم في عملها الحقيقي. سنكتب أولًا الدالة `report_uptime`:

```
report_uptime () {
        cat <<- _EOF_
                <H2>System Uptime</H2>
                <PRE>$(uptime)</PRE>
                _EOF_
        return
}
```

الدالة السابقة واضحة للغاية. لقد استخدمنا نمط إخراج here document لطباعة ترويسة القسم ومخرجات الأمر uptime محاطةً بوسم <pre> للحفاظ على تنسيق المخرجات. تُشبه الدالة report_disk_space الدالة السابقة:

```
report_disk_space () {
        cat <<- _EOF_
                <H2>Disk Space Utilization</H2>
                <PRE>$(df -h)</PRE>
                _EOF_
        return
}
```

تَستخدم الدالة الأمر df -h لتحديد مقدار المساحة التخزينية الفارغة من القرص. آخر دالة سنكتبها هي report_home_space:

```
report_home_space () {
        cat <<- _EOF_
                <H2>Home Space Utilization</H2>
                <PRE>$(du -sh /home/*)</PRE>
                _EOF_
        return
}
```

استخدمنا الأمر du مع الخيارَين sh-. لكن هذا ليس حلًّا كاملًا على الرغم من أن الدالة ستعمل على بعض الأنظمة (أوبنتو على سبيل المثال)، إلا أنها لن تعمل على البقية. السبب هو أن العديد من الأنظمة تضبط أذونات مجلد المنزل لمنع قراءته من قِبل باقي المستخدمين، وهو سبب مقنع هدفه الحفاظ على أمن بيانات المستخدمين. ستعمل دالة report_home_space على تلك الأنظمة إذا نُفِّذَ السكربت بامتياز الجذر. لكن الحل الأفضل هو جعل السكربت يُعدِّل سلوكه بالاعتماد على الأذونات المعطاة للمستخدم. وهذا هو موضوع

الفصل القادم.

دوال الشِل في ملف bashrc.

دوال الشِل هي بديل ممتاز عن الأوامر البديلة، وهي الطريقة المفضلة لإنشاء أوامر جديدة للاستخدام الشخصي. الأوامر البديلة محدودة جدًا حيث لا تدعم جميع أوامر أو ميزات الصدفة؛ على النقيض من ذلك، تسمح دوال الشِل بالقيام بجميع العمليات التي يمكن كتابتها كسكربت منفصل. على سبيل المثال، إذا أعجبتنا الدالة report_disk_space من سكربتنا السابق وأردنا إنشاء دالة لها الاسم "ds" في ملف ".bashrc":

```
ds () {
    echo "Disk Space Utilization For $HOSTNAME"
    df -h
}
```

الخلاصة

لقد تعرفنا في هذا الفصل على طريقة لتصميم البرامج تسمى Top-Down، وشاهدنا كيف نستخدم دوال الشِل لبناء المكونات الأساسية التي تعتمد عليها السكربتات. وتعلمنا أيضًا كيف يمكن استخدام المتغيرات المحلية لجعل الدوال مستقلّةً عن بعضها البعض. وهذا ما يسمح بكتابة دوال محمولة يمكن استخدامها في عدّة برامج، مما يوفر علينا وقتًا طويلًا.

الفصل السابع والعشرون:

بُنى التحكم: الدالة الشرطية if

لقد واجهنا مشكلةً في الفصل السابق: كيف نجعل برنامج توليد التقارير يَتَكيّف مع امتيازات المستخدم الذي يُشغله؟ يتطلب منا حلّ هذه المشكلة إيجاد طريقة "لتغيير الاتجاهات" في السكربت بالاعتماد على نتيجة اختبار محدد. أي أننا نريد -في المصطلحات التقنية- أن يتفرّع (branch) البرنامج.

لنفترض هذا المثال البسيط المكتوب في "أشباه الأكواد" (pseudo code)، أي محاكاة للغة البرمجة لإيصال فكرة البرنامج إلى البشر:

```
X=5
If X = 5, then:
        Say "X equals 5."
Otherwise:
        Say "X is not equal to 5."
```

هذا مثال عن التفرّع بالاعتماد على الشرط "هل `x = 5` ؟"، إذا كان ذاك الشرط محققًا فستُطبَع الجملة "`X equals 5.`"، عدا ذلك ستُطبَع العبارة "`X is not equal to 5.`".

if

إذا أردنا "تكويد" الفقرة السابقة (أي كتابة الكود المسؤول عنها) باستخدام الشِل:

```
x=5

if [ $x = 5 ]; then
        echo "x equals 5."
else
        echo "x does not equal 5."
fi
```

أو بكتابتها مباشرةً في سطر الأوامر (أقصر بكثير):

```
[me@linuxbox ~]$ x=5
```

```
[me@linuxbox ~]$ if [ $x = 5 ]; then echo "equals 5"; else echo "does not equal 5"; fi
equals 5
[me@linuxbox ~]$ x=0
[me@linuxbox ~]$ if [ $x = 5 ]; then echo "equals 5"; else echo "does not equal 5"; fi
does not equal 5
```

نفّذنا في المثال السابق الأمر مرّتَين: المرة الأولى عندما كانت قيمة المتغير x تساوي 5، حيث طُبِعَت العبارة "equals 5"؛ والمرّة الثانية عندما كانت قيمة المتغير x تساوي 0، وهذا ما أدى إلى طباعة العبارة "does not equal 5".

الشكل العام للدالة الشرطية if:

```
if commands; then
        commands
[elif commands; then
        commands...]
[else
        commands]
fi
```

حيث commands هي قائمة الأوامر. ربما يكون الأمر مربكًا للوهلة الأولى، لكن قبل أن نشرحه بالتفصيل، لنكتشف كيف تعرف الصدفة نجاح تنفيذ أمرٍ ما من عدمه.

حالة الخروج

تُرسِل الأوامر (بما فيها السكربتات ودوال الشِل التي نكتبها) قيمة إلى النظام عند انتهاء تنفيذها تسمى "حالة الخروج" (exit status). حالة الخروج، التي هي قيمة عددية تتراوح من 0 إلى 255، تُحَدِّد نجاح أو فشل تنفيذ أمرٍ ما. حسب ما يتداول، القيمة 0 تعني نجاح تنفيذ الأمر، بينما أيّة قيمة أُخرى تعني فشله. تُوفر الصدفة متغيرًا يمكننا استخدامه لمعرفة حالة الخروج للأمر الذي نُفِّذ سابقًا، كالآتي:

```
[me@linuxbox ~]$ ls -d /usr/bin
/usr/bin
[me@linuxbox ~]$ echo $?
```

```
0
[me@linuxbox ~]$ ls -d /bin/usr
ls: cannot access /bin/usr: No such file or directory
[me@linuxbox ~]$ echo $?
2
```

نفذنا، في المثال السابق، الأمر `ls` مرّتين. أول مرّة نُفِّذ الأمر فيها بنجاح؛ وإذا عرضنا قيمة المتغير `?$` فسنلاحظ أن الناتج هو 0. أما عند تنفيذنا للأمر `ls` على مجلد غير موجود، فإنه سيطبع رسالة خطأ وعند محاول عرض قيمة المتغير `?$` فسنحصل على القيمة 2، التي تشير إلى حدوث خطأ في تنفيذ الأمر. بعض الأوامر تستخدم حالة الخروج لتوفير معلوماتٍ عن الخطأ الذي واجهته، لكن مع ذلك، تستخدم العديد من الأوامر حالة الخروج ذات الرقم 1. تحتوي صفحات الدليل man عادةً على قسم ذي العنوان `"Exit Status"` يشرح حالات الخروج التي يستخدمها الأمر. تذكّر أن القيمة 0 تشير دائمًا إلى نجاح التنفيذ.

توفر الصدفة أمرَين مُضمنَين بسيطَين للغاية لا يقومان بأي شيءٍ سوى إعادة القيمة 0 أو 1 بعد انتهائهما. الأمر `true` يعيد 0 دائمًا، والأمر `false` يعيد 1 دائمًا.

```
[me@linuxbox ~]$ true
[me@linuxbox ~]$ echo $?
0
[me@linuxbox ~]$ false
[me@linuxbox ~]$ echo $?
1
```

بإمكاننا استخدام هذين الأمرَين للتعرّف على طريقة عمل عبارة `if`. ماذا تفعل العبارة `if` لكي تحدد نجاح الأوامر من عدمه؟

```
[me@linuxbox ~]$ if true; then echo "It's true."; fi
It's true.
[me@linuxbox ~]$ if false; then echo "It's true."; fi
[me@linuxbox ~]$
```

سيُنفذ الأمر `echo "It's true."` إذا نُفِّذَ الأمر الذي يتبع الكلمة `if` بنجاح، ولن ينفذ إذا لم ينفذ الأمر الذي يتبع الكلمة `if` بنجاح. إذا أُتبعَت الكلمة `if` بسلسلة من الأوامر، فستؤخذ بعين الاعتبار قيمة حالة الخروج لآخر أمر فقط:

```
[me@linuxbox ~]$ if false; true; then echo "It's true."; fi
```

```
It's true.
[me@linuxbox ~]$ if true; false; then echo "It's true."; fi
[me@linuxbox ~]$
```

test

أكثر أمر مستخدم مع بنية التحكم if هو "test". يقوم الأمر test بالعديد من الفحوصات والتحققات. يوجد له شكلين متكافئين:

```
test expression
```

والشكل الأشهر:

```
[ expression ]
```

حيث expression هو التعبير الذي سيحدد النتيجة هل هي true أم false. يعيد الأمر test حالة الخروج 0 إذا كان التعبير محققًا، وحالة خروج 1 عدا ذلك.

التعابير الخاصة بالملفات

تُستخدم التعابير الآتية لتحديد حالة الملفات:

الجدول 27-1: تعابير الملفات الخاصة بالأمر test

التعبير	يكون محققًا إذا كان
file1 -ef file2	الملفان file1 و file2 يشيران إلى نفس رقم العقدة (أي أن الملفين يشيران إلى نفس الملف عن طريق الوصلات الصلبة).
file1 -nt file2	الملف file1 أجدد من الملف file2.
file1 -ot file2	الملف file1 أقدم من الملف file2.
-b file	الملف file موجود وهو ملف جهاز كُتلي.
-c file	الملف file موجود وهو ملف جهاز محرفي.
-d file	file موجود وهو مجلد.
-e file	الملف file موجود.

-f file	الملف file موجود وهو ملف عادي.
-g file	الملف file موجود وقد حُدِّدَت خاصية set-group-ID.
-G file	الملف file موجود وهو مملوك لمجموعة المستخدم الحالي.
-k file	الملف file موجود وقد حددت الخاصية "sticky bit".
-L file	الملف file موجود وهو وصلة رمزية.
-O file	الملف file موجود وهو مملوك للمستخدم الحالي.
-p file	الملف file موجود وهو "أنبوبة مسماة" (named pipe).
-r file	الملف file موجود وهو قابل للقراءة (أي يوجد إذن القراءة للمستخدم الحالي).
-s file	الملف file موجود وحجمه التخزيني أكبر من الصفر.
-S file	الملف file موجود وهو مقبس شبكي (network socket).
-t fd	fd هو مقبض لملف موجه إلى/من الطرفية. يمكن استخدام هذا الاختبار لتحديد إذا ما أُعيد توجيه مجاري الدخل، أو الخرج، أو الخطأ القياسية.
-u file	الملف file موجود وحددت خاصية setuid.
-w file	الملف file موجود وهو قابل للكتابة من قِبل المستخدم الحالي.
-x file	الملف file موجود وهو قابل للتنفيذ من قِبل المستخدم الحالي.

يشرح السكربت الآتي مختلف تعابير الملفات:

```
#!/bin/bash

# test-file: Evaluate the status of a file

FILE=~/.bashrc

if [ -e "$FILE" ]; then
	if [ -f "$FILE" ]; then
```

```
		echo "$FILE is a regular file."
	fi
	if [ -d "$FILE" ]; then
		echo "$FILE is a directory."
	fi
	if [ -r "$FILE" ]; then
		echo "$FILE is readable."
	fi
	if [ -w "$FILE" ]; then
		echo "$FILE is writable."
	fi
	if [ -x "$FILE" ]; then
		echo "$FILE is executable/searchable."
	fi
else
	echo "$FILE does not exist"
	exit 1
fi

exit
```

يفحص السكربت السابق الملف الذي أُسند مساره إلى الثابت FILE ويُظهِر النتائج أثناء عملية الفحص. يوجد أمران مثيران للاهتمام يجب ملاحظتهما في هذا السكربت. أولًا، لاحظ أن المتغير $FILE قد تم "اقتباسه" في التعابير. لا يشترط فعل ذلك، لكننا استخدمناه كوقاية من كون الوسيط فارغًا. إذا تمت توسعة $FILE وحصلنا على قيمة فارغة، فسيؤدي ذلك إلى حدوث خطأ. نستطيع التأكد من وجود سلسلة نصية (وإن كانت فارغة) بعد المعامل باستخدام علامتي الاقتباس. لاحظ أيضًا وجود الأمر exit قرب نهاية السكربت. يقبل الأمر exit وسيطًا واحدًا اختياريًا يمثل حالة الخروج للسكربت. إذا لم يحدد ذاك الوسيط، فستُستخدم حالة الخروج لآخر أمرٍ مُنفَّذ. يسمح استخدام exit بهذه الطريقة للسكربت بأن يعلن فشل تنفيذه إذا تمت توسعة $FILE وحصل على مسار ملف غير موجود. وجود الأمر exit في نهاية السكربت هو عادة من عادات كتابة السكربتات. عندما يصل تنفيذ السكربت إلى آخره، فسيتم الانتهاء بحالة خروج آخر أمر مُنفَّذ.

وبشكلٍ مشابه، تعيد دوال الشِل حالة الخروج بتحديد وسيط رقمي إلى الأمر return. إذا أردنا تحويل السكربت السابق إلى دالة شِل لتضمينه في برنامج أكبر، فنستطيع استبدال الأمر exit بالأمر return:

```
test_file () {
```

```
        # test-file: Evaluate the status of a file

        FILE=~/.bashrc

        if [ -e "$FILE" ]; then
                if [ -f "$FILE" ]; then
                        echo "$FILE is a regular file."
                fi
                if [ -d "$FILE" ]; then
                        echo "$FILE is a directory."
                fi
                if [ -r "$FILE" ]; then
                        echo "$FILE is readable."
                fi
                if [ -w "$FILE" ]; then
                        echo "$FILE is writable."
                fi
                if [ -x "$FILE" ]; then
                        echo "$FILE is executable/searchable."
                fi
        else
                echo "$FILE does not exist"
                return 1
        fi
}
```

التعابير الخاصة بالسلاسل النصية

تُستخدم التعابير الآتية لتحديد حالة السلاسل النصية:

الجدول 27-2: التعابير النصية في test

التعبير	يكون محققًا إذا
`string`	كانت السلسلة النصية `string` غير معدومة (`null`).
`-n string`	كان طول السلسلة النصية `string` أكبر من الصفر.

-z string	كان طول السلسة النصية string مساويًا للصفر.
string1 = string2 string1 == string2	كانت السلسلتان النصيتان string1 و string2 متساويتين. وعلى الرغم من إمكانية استخدام علامة المساواة المفردة "="، إلا انه يُنصَح (وبشِدّة) باستخدام علامتي مساواة "==".
string1 != string2	كانت السلسلتان النصيتان غير متساويتين.
string1 > string2	كانت السلسلة string1 تأتي قبل string2 عند ترتيبهما (sort).
string1 < string2	كانت السلسلة string1 تأتي بعد string2 عند ترتيبهما (sort).

تحذير: يجب وضع المعاملين ">" و "<" بين علامتي اقتباس (أو تهريبهما باستخدام الشرطة المائلة الخلفية) عند استخدامهما مع test. إذا لم يتم ذلك، فسيُفسران على أنهما معاملي إعادة التوجيه، مما قد يسبب نتائج كارثية. لاحظ أيضًا أنه وعلى الرغم من أن توثيق bash يذكر أن نمط الترتيب المستخدم هو نفسه نمط الترتيب المحدد في المحليّة (locale)، إلا أنه ليس كذلك! سيُستخدم نمط ترتيب ASCII (POSIX) في جميع إصدارات bash بما فيها الإصدار 4.0.

يوضح السكربت الآتي استخدام التعابير الخاصة بالسلاسل النصية:

```
#!/bin/bash

# test-string: evaluate the value of a string

ANSWER=maybe

if [ -z "$ANSWER" ]; then
        echo "There is no answer." >&2
        exit 1
fi
if [ "$ANSWER" = "yes" ]; then
       echo "The answer is YES."
elif [ "$ANSWER" = "no" ]; then
       echo "The answer is NO."
elif [ "$ANSWER" = "maybe" ]; then
```

```
        echo "The answer is MAYBE."
else
        echo "The answer is UNKNOWN."
fi
```

حددنا، في المثال السابق، حالة الثابت ANSWER. حددنا أولًا إذا كانت السلسلة النصية فارغة. إذا كانت كذلك، فسينتهي السكربت بحالة خروج تساوي 1. لاحظ إعادة التوجيه المُطبَّقة على الأمر echo، حيث ستُرسَل رسالة الخطأ "`There is no answer.`" إلى مجرى الخطأ القياسي؛ وهذا ما يجب فعله مع رسائل الخطأ. إذا لم تكن السلسلة النصية فارغة، فسنختبر قيمتها إذا كانت "yes"، أو "no"، أو "maybe" وذلك باستخدام `elif` التي هي اختصار للعبارة "`else if`" (أي "إذا لم يتحقق الشرط الأول فجرّب الشرط الآتي." وهكذا). يمكننا إنشاء اختبارات معقدة أكثر باستخدام `elif`.

التعابير الخاصة بالأرقام

يمكن استخدام التعابير الآتية مع الأرقام:

الجدول 27-3: التعابير العددية في test

التعبير	يكون محققًا إذا كان
`integer1 -eq integer2`	`integer1` و `integer2` متساويين.
`integer1 -ne integer2`	`integer1` و `integer2` غير متساويين.
`integer1 -le integer2`	`integer1` أقل أو يساوي `integer2`.
`integer1 -lt integer2`	`integer1` أقل من `integer2`.
`integer1 -ge integer2`	`integer1` أكبر أو يساوي `integer2`.
`integer1 -gt integer2`	`integer1` أكبر من `integer2`.

يشرح السكربت الآتي استخدامها:

```
#!/bin/bash

# test-integer: evaluate the value of an integer.

INT=-5
```

```
if [ -z "$INT" ]; then
       echo "INT is empty." >&2
       exit 1
fi
if [ $INT -eq 0 ]; then
       echo "INT is zero."
else
       if [ $INT -lt 0 ]; then
               echo "INT is negative."
       else
               echo "INT is positive."
       fi
       if [ $((INT % 2)) -eq 0 ]; then
               echo "INT is even."
       else
               echo "INT is odd."
       fi
fi
```

الجزء المثير للاهتمام من السكربت هو طريقة تحديد فيما إذا كان العدد زوجيًا أم فرديًا. وذلك بمعاينة باقي القسمة على العدد 2؛ حيث يكون زوجيًا إذا كان باقي القسمة 0، وفرديًا فيما عدا ذلك.

نسخةٌ أكثر حداثةً من test

تحتوي الإصدارات الحديثة من bash أمرًا مركبًا جديدًا يُمثِّل بديلًا محسّنًا من test له الشكل العام الآتي:

```
[[ expression ]]
```

وكما في test، فإن expression هو التعبير الذي سيتم التحقق فيما إذا كان صحيحًا (true) أم لا (false). الأمر [[ ]] شبيه جدًا بالأمر test حيث يدعم جميع تعابيره، لكنه يضيف تعبيرًا مفيدًا جدًا خاصًا بالسلاسل النصية:

```
string1 =~ regex
```

الذي يكون محققًا إذا تمت مطابقة السلسلة النصية string بنمط التعابير النظامية الموسعة regex. وهذا ما يفتح المجال أمام العديد من الإمكانيات للقيام بمهام كالتحقق من صحة البيانات... في مثالنا السابق عن

التعابير العددية، سيفشل تنفيذ السكربت إذا حوى الثابت INT على أي شيء عدا الأرقام. سيحتاج السكربت إلى طريقة للتحقق من وجود رقم مُخزَّن في ذاك الثابت. بإمكاننا تحسين ذاك السكربت باستخدام [[ ]] مع المعامل "=~":

```
#!/bin/bash

# test-integer2: evaluate the value of an integer.

INT=-5

if [[ "$INT" =~ ^-?[0-9]+$ ]]; then
        if [ $INT -eq 0 ]; then
                echo "INT is zero."
        else
                if [ $INT -lt 0 ]; then
                      echo "INT is negative."
                else
                      echo "INT is positive."
                fi
                if [ $((INT % 2)) -eq 0 ]; then
                      echo "INT is even."
                else
                      echo "INT is odd."
                fi
        fi
else
        echo "INT is not an integer." >&2
        exit 1
fi
```

تأكدنا من أن قيمة الثابت int تحتوي على سلسلة نصية التي تبدأ اختياريًا بإشارة السالب يتبعها رقم واحد أو أكثر. لاحظ أيضًا أن التعبير لا يسمح بالقيم الفارغة.

ميزة أُخرى للأمر [[ ]] هي دعم المعامل == لمطابقة التعابير النظامية بشكلٍ مشابه لتوسعة أسماء الملفات. على سبيل المثال:

```
[me@linuxbox ~]$ FILE=foo.bar
```

```
[me@linuxbox ~]$ if [[ $FILE == foo.* ]]; then
> echo "$FILE matches pattern 'foo.*'"
> fi
foo.bar matches pattern 'foo.*'
```

وهذا ما يجعل [[ ]] مفيدًا للتحقق من المسارات.

(( )) مصمّم خصيصًا للأرقام

توفر bash -بالإضافة إلى [[ ]]- الأمر المركب (( )) المفيد في إجراء العمليات على الأرقام؛ حيث يدعم جميع العمليات الحسابية، الموضوع الذي سنتحدث عنه بالتفصيل في الفصل 34.

يكون الأمر (( )) محققًا إذا كان ناتج العملية الحسابية أي عدد لا يساوي الصفر.

```
[me@linuxbox ~]$ if ((1)); then echo "It is true."; fi
It is true.
[me@linuxbox ~]$ if ((0)); then echo "It is true."; fi
[me@linuxbox ~]$
```

يمكننا تبسيط سكربت test-integer2 كثيرًا باستخدام (( )) كالآتي:

```
#!/bin/bash

# test-integer2a: evaluate the value of an integer.

INT=-5

if [[ "$INT" =~ ^-?[0-9]+$ ]]; then
        if ((INT == 0)); then
                echo "INT is zero."
        else
                if ((INT < 0)); then
                        echo "INT is negative."
                else
                        echo "INT is positive."
                fi
                if (( ((INT % 2)) == 0)); then
```

```
                        echo "INT is even."
                else
                        echo "INT is odd."
                fi
        fi
else
        echo "INT is not an integer." >&2
        exit 1
fi
```

لاحظ استخدامنا المعاملات: "أكبر-من" و "أصغر-من" و "يساوي" للتحقق من المساواة. لاحظ أيضًا أن الأمر المركب (( )) هو أمر مُضمَّن في الصدفة وليس أمرًا خارجيًا، فيستطيع الوصول إلى قيم المتغيرات بأسمائها دون الحاجة إلى القيام بعملية توسعة؛ سنناقش (( )) وباقي العمليات الحسابية بالتفصيل في الفصل 34.

التعابير المركبة

من الممكن أيضًا دمج أكثر من تعبير لإنشاء اختبارات معقدة. تُدمَج التعابير عن طريق المعاملات المنطقية. شاهدنا هذه المعاملات سابقًا في الفصل 17 عندما ناقشنا الأمر find. توجد ثلاث عمليات منطقية تُستخدم مع test و [[ ]] هي: AND، و OR، و NOT. يستخدم test و [[ ]] معاملات مختلفة لتمثيل تلك العمليات:

الجدول 27-4: المعاملات المنطقية

العملية	الأمر test	[[ ]] و (( ))
AND	-a	&&
OR	-o	||
NOT	!	!

هنا مثال عن استخدام العملية AND. يُحدِّد السكربت الآتي فيما إذا كان رقمٌ ما موجودًا في مجال محدد من القيم:

```
#!/bin/bash

# test-integer3: determine if an integer is within a
# specified range of values.
```

```
MIN_VAL=1
MAX_VAL=100

INT=50

if [[ "$INT" =~ ^-?[0-9]+$ ]]; then
	if [[ INT -ge MIN_VAL && INT -le MAX_VAL ]]; then
		echo "$INT is within $MIN_VAL to $MAX_VAL."
	else
		echo "$INT is out of range."
	fi
else
		echo "INT is not an integer." >&2
		exit 1
fi
```

تأكدّنا في هذا السكربت أن قيمة العدد INT تكون بين القيمتَين MIN_VAL و MAX_VAL. وذلك باستخدام الأمر [[ ]] الذي يحتوي على تعبيرَين مفصولَين بمعامل &&. بإمكاننا أيضًا كتابة الكود السابق باستخدام test كما يلي:

```
	if [ $INT -ge $MIN_VAL -a $INT -le $MAX_VAL ]; then
		echo "$INT is within $MIN_VAL to $MAX_VAL."
	else
		echo "$INT is out of range."
	fi
```

يعكس معامل النفي "!" ناتج التعبير. حيث يعيد true إذا كان التعبير false، ويُعيد false إذا كان التعبير true. سنُعدِّل السكربت السابق لإيجاد قيم INT التي هي خارج المجال المحدد:

```
#!/bin/bash

# test-integer4: determine if an integer is outside a
# specified range of values.

MIN_VAL=1
MAX_VAL=100
```

```
INT=50

if [[ "$INT" =~ ^-?[0-9]+$ ]]; then
	if [[ ! (INT -ge MIN_VAL && INT -le MAX_VAL) ]]; then
		echo "$INT is outside $MIN_VAL to $MAX_VAL."
	else
		echo "$INT is in range."
	fi
else
	   echo "INT is not an integer." >&2
	   exit 1
fi
```

وضعنا أقواسًا حول التعبير لجعله مجموعة واحدة. لأننا لو وضعنا معامل النفي دون أقواس، فسيتم نفي أول تعبير فقط. يمكننا إعادة كتابة التعبير السابق باستخدام `test` بالشكل الآتي:

```
	if [ ! \( $INT -ge $MIN_VAL -a $INT -le $MAX_VAL \) ]; then
		echo "$INT is outside $MIN_VAL to $MAX_VAL."
	else
		echo "$INT is in range."
	fi
```

لما كانت جميع التعابير والمعاملات التي يستخدمها `test` تُفسَّر على أنها وسائط للأمر `test` بواسطة الصدفة (على النقيض من `[[ ]]` و `(( ))`)، فيجب أن تُقتبس المحارف التي لها معنى خاص للصدفة مثل < و > و (و)؛ أو تُهرّب.

ولأن `test` و `[[ ]]` يقومان تقريبًا بنفس العمل، فأيهما افضل؟ الأمر `test` هو تقليدي (وجزء من POSIX)، بينما الأمر المركب `[[ ]]` هو خاص بالصدفة bash. من المهم معرفة طريقة عمل `test` لأنه واسع الانتشار، لكن الأمر المركب `[[ ]]` أسهل بكثير في الكتابة.

المحموليّة هي فزاعة ذوي العقول الصغيرة

إذا تحدثت إلى مستخدمي يونكس "الحقيقيين"، ستكتشف بسرعة أن العديد منهم لا يحب ليُنكس كثيرًا. يقولون أن السبب هو "عدم وضوحه". مبدأ أساسي من مبادئ أتباع يونكس هو أن كل شيء

يجب أن يكون "محمولًا". هذا يعني أن أيّ سكربت تكتبه يجب أن يكون يعمل، دون أيّ تعديل، على أي نظام شبيه بِيونكس.

لدى أتباع يونكس سبب وجيه للاعتقاد بذلك. لأنهم شاهدوا ما فعلته الإضافات التجارية إلى الأوامر والصدفات على عالم يونكس قبل وجود معيار POSIX، لذا، فإنهم قلقون قلقًا طبيعيًّا من تأثير لينُكس على نظامهم المحبب.

لكن للمحمولية جانب سلبي كبير: إنها توقف التطور! إنها تتطلب أن يتم القيام بالأشياء باستخدام "القاسم المشترك الأصغر" للتقنيات. في حالة برمجة الشِل، هذا يعني كتابة جميع السكربتات بشكل متوافق مع صدفة Bourne الأصلية: sh.

ذاك الجانب السلبي هو عذر الشركات التجارية التي تستخدمه لاستحقاق ثمن إضافاتهم التجارية، لكنهم يسمونها "ابتكارات". لكنهم في الواقع يحاولون جعل زبائنهم دائمين.

لا تحتوي أدوات غنو، كالصدفة bash، على أيّة قيود. إنهم يدعمون المحمولية بإتباع المعايير القياسية وتوفير البرمجيات للجميع. بإمكانك تثبيت bash وغيرها من أدوات غنو على أي نظام تشغيل تقريبًا، وحتى ويندوز، مجانًا. لذا استخدم بكل حرية جميع ميزات bash. هذه هي المحمولية الحقيقية!

معاملات التحكم: طريقة أُخرى للتفرّع

تُوفر الصدفة bash معاملَيّ تحكم يُستخدمان للتفرع. المعاملين && (AND) و || (OR) يُعملان بنفس آلية المعاملات المنطقية في الأمر المركب `[[ ]]`. الشكل العام هو:

```
command1 && command2
```

و:

```
command1 || command2
```

من المهم فهم سلوك هذين المعاملين. عند استخدام المعامل &&، فإن command1 سينفَّذ و command2 سينفَّذ فقط إذا نُفِّذ الأمر command1 بنجاح. وعند استخدام المعامل ||، فإن command1 سينفَّذ و command2 سينفَّذ فقط إذا لم يُنفَّذ الأمر command1 بنجاح.

في الحياة العملية، هذا يعني أننا نستطيع القيام بشيء شبيه بالآتي:

```
[me@linuxbox ~]$ mkdir temp && cd temp
```

سيُنشئ الأمر السابق المجلد المسمى temp، وفي حال أُنشئ بنجاح، فسيُغيّر مجلد العمل الحالي إلى temp.

الأمر الثاني سيُنفَّذ في حال نجح تنفيذ الأمر الأول (mkdir). وبآلية مشابهة، فإن أمرًا كهذا الأمر:

```
[me@linuxbox ~]$ [ -d temp ] || mkdir temp
```

سيختبر وجود المجلد temp، فإذا فشل الاختبار، فسيُنشئ ذاك المجلد. هذه الآلية مفيدة جدًا في السكربتات لمعالجة الأخطاء، وهو موضوع سنناقشه في فصولٍ لاحقة. على سبيل المثال، يمكننا إضافة هذا السطر في سكربت:

```
[ -d temp ] || exit 1
```

إذا تطلب السكربت وجود المجلد temp، وفي حال عدم وجوده، فسينتهي تنفيذ السكربت بحالة خروج تساوي 1.

الخلاصة

بدأنا هذا الفصل بسؤال: كيف نستطيع جعل سكربت sys_info_page يتأقلم مع امتيازات المستخدم؟ بعد تعرّفنا على if، نستطيع حلّ هذه المشكلة بتعديل كود الدالة report_home_space:

```
report_home_space () {
        if [[ $(id -u) -eq 0 ]]; then
                cat <<- _EOF_
                        <H2>Home Space Utilization (All Users)</H2>
                        <PRE>$(du -sh /home/*)</PRE>
                        _EOF_
        else
                cat <<- _EOF_
                        <H2>Home Space Utilization ($USER)</H2>
                        <PRE>$(du -sh $HOME)</PRE>
                        _EOF_
        fi
        return
}
```

تحققنا من ناتج الأمر id. سيطبع الأمر id -عند استخدام الخيار "u-"- رقم هوية المستخدم الحالي. يكون رقم هوية المستخدم الجذر دائمًا يساوي الصفر. ورقم هوية أي مستخدم آخر هو أكبر من الصفر. بمعرفة ذلك، أنشأنا نسختَين من المستند، واحدة تظهر عند تنفيذ السكربت بامتيازات الجذر، وواحدة تظهر عند تنفيذ

السكربت بامتيازات المستخدم العادي.

سنأخذ الآن استراحة من برنامج sys_info_page. لكن لا تقلق، سنعود إليه لاحقًا. إلى ذاك الحين، سنشرح بعض الأمور التي سنحتاجها لمتابعة عملنا.

الفصل الثامن والعشرون:

قراءة مدخلات لوحة المفاتيح

لا توفر جميع السكربتات التي كتبناها إلى الآن ميزة شائعة، إلا وهي التفاعلية. أي قدرة البرنامج على التواصل مع المستخدم. وعلى الرغم من أن العديد من البرامج لا تحتاج إلى أن تكون تفاعلية، إلا أن بعضها الآخر يحقق فائدةً كبيرةً من ذلك. لنضرب مثلًا هذا السكربت من الفصل الماضي:

```
#!/bin/bash

# test-integer2: evaluate the value of an integer.

INT=-5

if [[ "$INT" =~ ^-?[0-9]+$ ]]; then
	if [ $INT -eq 0 ]; then
		echo "INT is zero."
	else
		if [ $INT -lt 0 ]; then
			echo "INT is negative."
		else
			echo "INT is positive."
		fi
		if [ $((INT % 2)) -eq 0 ]; then
			echo "INT is even."
		else
			echo "INT is odd."
		fi
	fi
else
	echo "INT is not an integer." >&2
	exit 1
fi
```

في كل مرّة نريد تغيير قيمة INT فيها، سنحتاج إلى تعديل السكربت. سيكون من المفيد هنا جعل

السكربت يسأل المستخدم عن القيمة. سنتعرف في هذا الفصل على طريقة إضافة التفاعلية إلى برامجنا.

قراءة القيم من مجرى الدخل القياسي باستخدام read

يُستخدم الأمر read المضمن في الصدفة لقراءة سطر واحد من المدخلات من مجرى الدخل القياسي. يُستخدم هذا الأمر لقراءة مدخلات لوحة المفاتيح، أو قراءة سطر من البيانات الموجودة في ملف إذا أُستخدمت إعادة التوجيه. الشكل العام للأمر هو:

```
read [-options] [variable...]
```

حيث options هو خيار واحد أو أكثر من الخيارات التي سيأتي ذكرها في الجدول أدناه، و variable هو اسم متغير واحد أو أكثر الذي ستُخزَّن فيه المدخلات. إذا لم يُحدَّد اسم المتغير، فسيُخزَّن سطر المدخلات في متغير الصدفة REPLY.

افتراضيًّا، يُسنِد read قيمة الحقول التي قرأها من مجرى الدخل القياسي إلى المتغيرات المحددة. إذا عدّلنا في السكربت السابق كي يستخدم read، فسوف يصبح شبيهًا بالآتي:

```
#!/bin/bash

# read-integer: evaluate the value of an integer.

echo -n "Please Enter an integer -> "
read int

if [[ "$int" =~ ^-?[0-9]+$ ]]; then
	if [ $int -eq 0 ]; then
		echo "$int is zero."
	else
		if [ $int -lt 0 ]; then
		      echo "$int is negative."
		else
		      echo "$int is positive."
		fi
		if [ $((int % 2)) -eq 0 ]; then
		      echo "$int is even."
		else
		      echo "$int is odd."
```

```
                    fi
            fi
else
            echo "Input value is not an integer." >&2
            exit 1
fi
```

استخدمنا الأمر echo مع الخيار n- كي لا يَطبع محرف السطر الجديد بعد طباعة العبارة "Please Enter an integer ->"، ثم استخدمنا read لقراءة القيمة العددية وتخزينها في المتغير int. يؤدي تشغيل السكربت إلى إظهار النتائج الآتية:

```
[me@linuxbox ~]$ read-integer
Please Enter an integer -> 5
5 is positive.
5 is odd.
```

يمكن أن يُسنِد read المدخلات إلى أكثر من متغير، كما في السكربت الآتي:

```
#!/bin/bash

# read-multiple: read multiple values from keyboard

echo -n "Enter one or more values > "
read var1 var2 var3 var4 var5

echo   "var1   = '$var1'"
echo   "var2   = '$var2'"
echo   "var3   = '$var3'"
echo   "var4   = '$var4'"
echo   "var5   = '$var5'"
```

أسندنا (وأظهرنا) في المثال السابق خمس قيم. لاحظ سلوك read عند إعطاء عدد مختلف من القيم:

```
[me@linuxbox ~]$ read-multiple
Enter one or more values > a b c d e
var1 = 'a'
var2 = 'b'
```

```
var3 = 'c'
var4 = 'd'
var5 = 'e'
[me@linuxbox ~]$ read-multiple
Enter one or more values > a
var1 = 'a'
var2 = ''
var3 = ''
var4 = ''
var5 = ''
[me@linuxbox ~]$ read-multiple
Enter one or more values > a b c d e f g
var1 = 'a'
var2 = 'b'
var3 = 'c'
var4 = 'd'
var5 = 'e f g'
```

إذا استقبل read مدخلات أقل من العدد المتوقع، فستكون المتغيرات الإضافية فارغةً. أما إذا استقبل مدخلات أكثر من العدد المتوقع، فسيحتوي آخر متغير على أيّة مدخلات إضافية.

إذا لم تحدد وسائط للأمر read، فستُسند جميع المدخلات إلى متغير الصدفة REPLY:

```
#!/bin/bash

# read-single: read multiple values into default variable

echo -n "Enter one or more values > "
read

echo "REPLY = '$REPLY'"
```

يؤدي تشغيل السكربت السابق إلى إظهار النتائج الآتية:

```
[me@linuxbox ~]$ read-single
Enter one or more values > a b c d
REPLY = 'a b c d'
```

الخيارات

يدعم read الخيارات الآتية:

الجدول 28-1: خيارات read

الخيار	الشرح
`-a array`	إسناد المدخلات إلى المصفوفة array، ابتداءً من المفتاح 0. سنناقش المصفوفات في الفصل 35.
`-d delimiter`	استخدام أول محرف في السلسلة النصية delimiter للإشارة إلى نهاية المدخلات، بدلًا من محرف السطر الجديد.
`-e`	استخدام مكتبة Readline. هذا يسمح بتعديل المدخلات بنفس آلية التعديل في سطر الأوامر.
`-i string`	استخدم السلسلة النصية string كجواب افتراضي إذا ضغط المستخدم على Enter دون إدخال أيّ شيء. يتطلب هذا الخيار استخدام الخيار e-.
`-n num`	قراءة num محرفًا من المدخلات، عوضًا عن قراءة السطر بأكمله.
`-p prompt`	إظهار مِحث للإدخال يحتوي على السلسلة النصية prompt.
`-r`	وضع الإدخال "الخام" (raw). لن تُعامَل الشرطة المائلة الخلفية كمحرف هروب.
`-s`	وضع الإدخال الصامت. لن تظهر المحارف المُدخَلة على الشاشة أثناء كتابتها. هذا الخيار مفيد عند الطلب من المستخدم إدخال كلمة مرور أو غيرها من البيانات السرية.
`-t seconds`	تحديد المهلة الزمنية. إنهاء عملية الإدخال بعد seconds ثانية، يعيد read في هذه الحالة حالة خروج لا تساوي الصفر.
`-u fd`	قراءة المدخلات من الملف fd بدلًا من مجرى الدخل القياسي.

يمكننا القيام بالعديد من الأمور الممتعة مع read باستخدام الخيارات المختلفة. على سبيل المثال، يمكننا تحديد عبارة المِحث باستخدام الخيار "p-":

```
#!/bin/bash
```

```
# read-single: read multiple values into default variable

read -p "Enter one or more values > "

echo "REPLY = '$REPLY'"
```

وباستخدام الخيارَين ‎-t‎ و ‎-s‎، نستطيع إنشاء سكربت يقرأ المدخلات "السرية" وتنتهي المهلة الزمنية إذا لم يدخل المستخدم البيانات في الوقت المحدد:

```
#!/bin/bash

# read-secret: input a secret pass phrase

if read -t 10 -sp "Enter secret pass phrase > " secret_pass; then
        echo -e "\nSecret pass phrase = '$secret_pass'"
else
        echo -e "\nInput timed out" >&2
        exit 1
fi
```

يطلب هذا السكربت من المستخدم إدخال عبارة "سرية" وينتظر 10 ثوانٍ لذلك. إذا لم تُدخَل العبارة في الوقت المحدد، فسينتهي تنفيذ السكربت مع إظهار رسالة خطأ. ولن تظهر حروف الجملة على الشاشة أثناء كتابتها بسبب استخدام الخيار ‎-s‎.

من الممكن أيضًا توفير عبارة افتراضية كجواب باستخدام الخيارَين ‎-e‎ و ‎-i‎:

```
#!/bin/bash

# read-default: supply a default value if user presses Enter key.

read -e -p "What is your user name? " -i $USER
echo "You answered: '$REPLY'"
```

لقد طلبنا في هذا المثال من المستخدم إدخال اسمه واستخدمنا متغير البيئة USER لتوفير قيمة افتراضية. سيسند read المدخلات (أو القيمة الافتراضية إذا لم يعدلها المستخدم) إلى المتغير REPLY:

```
[me@linuxbox ~]$ read-default
What is your user name? me
You answered: 'me'
```

IFS

تُقطِّع الصدفة الكلمات في المدخلات التي تُقرأ بواسطة read. وكما لاحظنا، تُعامَل الكلمات المفصولة بفراغٍ واحدٍ أو أكثر على أنها عناصر منفصلة في سطر المدخلات، وستُسند إلى عدّة متغيرات بواسطة read. يمكن تغيير هذا السلوك عن طريق متغير الصدفة IFS (Internal Field Separator أي الفاصل الداخلي للحقول). تحتوي القيمة الافتراضية للمتغير IFS -التي تفصل ما بين العناصر- على فراغ، ومسافة جدولة، ومحرف السطر الجديد.

بإمكاننا تعديل قيمة IFS لتغيير الفاصل ما بين حقول المدخلات التي سيقرأها الأمر read. على سبيل المثال، يحتوي ملف /etc/passwd على أسطر من البيانات تستخدم النقطتين الرأسيتين فاصلًا ما بين الحقول. بإمكاننا جعل read يقرأ محتويات ملف /etc/passwd قراءةً صحيحةً ويُخزِّن كل حقل في متغير بإسناد ":" إلى المتغير IFS.

السكربت الآتي يقوم بذلك:

```
#!/bin/bash

# read-ifs: read fields from a file

FILE=/etc/passwd

read -p "Enter a user name > " user_name

file_info=$(grep "^$user_name:" $FILE)

if [ -n "$file_info" ]; then
	IFS=":" read user pw uid gid name home shell <<< "$file_info"
	echo "User =		'$user'"
	echo "UID =		'$uid'"
	echo "GID =		'$gid'"
	echo "Full Name = '$name'"
	echo "Home Dir. = '$home'"
```

```
	echo "Shell =		'$shell'"
else
	echo "No such user '$user_name'" >&2
	exit 1
fi
```

يطلب السكربت من المستخدم إدخال اسم أحد مُستخدمي النظام، ثم يُظهِر مختلف الحقول الموجودة في أحد الأسطر الذي يسجل بيانات المستخدم ويخزنها في ملف /etc/passwd. يحتوي السكربت السابق على سطرين مثيرين للاهتمام. الأول هو:

```
file_info=$(grep "^$user_name:" $FILE)
```

يسند هذا السطر مخرجات الأمر grep إلى المتغير file_info. التعبير النظامي المُستخدم من قِبل grep يضمن أن اسم المستخدم سيُطابِق سطرًا واحدًا فقط من ملف /etc/passwd.

السطر الثاني هو:

```
IFS=":" read user pw uid gid name home shell <<< "$file_info"
```

هذا السطر يحتوي على ثلاثة أقسام: عملية إسناد قيمة إلى متغير، والأمر read مع قائمة بأسماء المتغيرات ممررة كوسيط، ومعامل إعادة توجيه ذي شكلٍ غريب. سنناقش عملية إسناد القيمة إلى المتغير أولًا.

تسمح الصدفة بعملية إسناد القيم إلى المتغيرات قبل الأمر مباشرةً. عملية الإسناد تلك تؤدي إلى تغيير "البيئة" للأمر الذي يتبعها. أي أن تأثير عملية الإسناد مؤقت؛ فستُغيَّر البيئة خلال تنفيذ الأمر فقط. أسندنا القيمة ":" إلى المتغير IFS في المثال السابق. يمكننا أيضًا كتابتها بطريقة أُخرى كالآتي:

```
OLD_IFS="$IFS"
IFS=":"
read user pw uid gid name home shell <<< "$file_info"
IFS="$OLD_IFS"
```

خزّنا قيمة IFS، ثم أسندنا له قيمة جديدة، نفذنا الأمر read، وفي النهاية أعدنا القيمة القديمة للمتغير IFS. كما هو واضح، إن وضع المتغير قبل الأمر مباشرةً هو أسهل بكثير.

يشير المعامل "<<<" إلى ما يسمى "here string". here string تشبه إلى حدٍ كبير here document، لكنها أقصر وتحتوي على سلسلة نصية واحدة. في مثالنا السابق، قمنا بتمرير سطر من البيانات الموجودة في ملف /etc/passwd إلى الأمر read. ربما تتساءل عن سبب استخدمنا لهذه الطريقة الغريبة بدلًا من استخدام الأنابيب:

```
echo "$file_info" | IFS=":" read user pw uid gid name home shell
```

حسنًا، يوجد هنالك سبب...

لماذا لا تستطيع استخدام الأنابيب مع read

على الرغم من أن الأمر read يقرأ المُدخلات افتراضيًا من مجرى الدخل القياسي، إلا أننا لا نستطيع القيام بالآتي:

```
echo "foo" | read
```

توقعنا أن يعمل الأمر السابق بنجاح، لكنه لم يَكُن كذلك! سيبدو أن الأمر قد نُفِّذ بنجاح لكن المتغير REPLY سيكون فارغًا. لماذا؟

السبب يَكمن في طريقة تعامل الصدفة مع الأنابيب. في bash (وغيرها من الصدفات كصدفة sh)، تُنشئ الأنابيب "صدفات فرعية" (subshells). التي هي نسخة من الصدفة والبيئة المستخدمة، للقيام بتنفيذ الأوامر في الأنبوب. نُفِّذ الأمر read، في المثال السابق، في صدفة فرعية.

الصدفات الفرعية في الأنظمة الشبيهة بِيونكس تُنشئ نسخًا من البيئة للعمليات عند تنفيذها. "ستُدمَّر" نسخة البيئة في تلك العملية عند انتهاء تنفيذها. هذا يعني أن الصدفة الفرعية لن تُغيِّر في بيئة "العملية الأب". يسند read المتغيرات التي تكون جزءًا من البيئة. في المثال السابق، أسندنا القيمة "foo" إلى المتغير REPLY في بيئة الصدفة الفرعية؛ لكن عندما ينتهي تنفيذ الأمر فستُدمَّر الصدفة الفرعية وبيئتها، ولن نستطيع الاستفادة من عملية الإسناد.

استخدام here string هو إحدى الطرق التي تستطيع الالتفاف على هذه الإشكالية. سنناقش الطريقة الثانية في الفصل 36.

التحقق من صحة المدخلات

سنواجه تحديًا برمجيًا جديدًا بعد أن تعلمنا كيفية استقبال المدخلات من المستخدم الذي هو "التحقق من صحة المدخلات". أحد أهم الفروقات ما بين البرامج المكتوبة كتابةً جيدةً وتلك المكتوبة كتابةً سيئةً هو قابليّة تعامل البرنامج مع الأشياء غير المتوقعة. لحسن الحظ، الأشياء غير المتوقعة تكون عادةً على شكل مدخلات غير صحيحة. لقد تحققنا من المدخلات في أمثلة الفصل السابق، حيث تحققنا أن المدخلات هي عددية واستبعدنا ما عدا ذلك من مدخلات فارغة وغير عددية. من المهم أن تُجرى تلك الفحوصات في كل مرة يستقبل فيها البرنامج مدخلات من المستخدم لحمايته من المدخلات غير الصحيحة. هذا الأمر مهم جدًا خصوصًا للبرامج المشتركة ما بين العديد من المستخدمين. لا بأس في عدم التحقق من المدخلات إذا كان البرنامج سيُشغَّل لمرة واحدة ومن قِبل مبرمجه. لكن مع ذلك، من الجيد أن يتم التحقق من المدخلات إذا كان

البرنامج يقوم بأعمال "خطرة" كحذف الملفات.

هذا مثال عن برنامج يتحقق من مختلف أنواع المدخلات:

```
#!/bin/bash

# read-validate: validate input

invalid_input () {
        echo "Invalid input '$REPLY'" >&2
        exit 1
}

read -p "Enter a single item > "

# input is empty (invalid)
[[ -z $REPLY ]] && invalid_input

# input is multiple items (invalid)
(( $(echo $REPLY | wc -w) > 1 )) && invalid_input

# is input a valid filename?
if [[ $REPLY =~ ^[-[:alnum:]\._]+$ ]]; then
         echo "'$REPLY' is a valid filename."
         if [[ -e $REPLY ]]; then
                echo "And file '$REPLY' exists."
         else
                echo "However, file '$REPLY' does not exist."
         fi
         # is input a floating point number?
         if [[ $REPLY =~ ^-?[[:digit:]]*\.[[:digit:]]+$ ]]; then
                echo "'$REPLY' is a floating point number."
         else
                echo "'$REPLY' is not a floating point number."
         fi
         # is input an integer?
         if [[ $REPLY =~ ^-?[[:digit:]]+$ ]]; then
```

```
                echo "'$REPLY' is an integer."
        else
                echo "'$REPLY' is not an integer."
        fi
 else
        echo "The string '$REPLY' is not a valid filename."
fi
```

يطلب السكربت السابق من المستخدم إدخال أيّة قيمة. ثم ستُحلّل تلك القيمة لتحديد نوعها. وكما لاحظنا، يستخدم السكربت العديد من البُنى والأوامر التي تعلمناها إلى الآن. بما فيها دوال الشِل، و `[[ ]]`، و `(( ))`، ومعامل التحكم &&، و `if`، بالإضافة إلى استخدام مُكثَّف للتعابير النظامية.

القوائم

طريقة مشهورة لتحقيق التفاعلية في البرامج هي استخدام القوائم، تُظهِر البرامج التي تستخدم القوائم للمستخدم عدّة خيارات وتسأله أن يختار أحدها. على سبيل المثال، يمكننا تخيل أن تلك البرامج تُظهِر قائمةً شبيهةً بالقائمة الآتية:

```
Please Select:

1. Display System Information
2. Display Disk Space
3. Display Home Space Utilization
0. Quit

Enter selection [0-3] >
```

يمكننا تطوير برنامج sys_info_page لكي نُضمِّن فيه قائمة تسأل المستخدم عن المعلومات التي يريد إظهارها:

```
#!/bin/bash

# read-menu: a menu driven system information program

clear
echo "
```

```
Please Select:

1. Display System Information
2. Display Disk Space
3. Display Home Space Utilization
0. Quit
"
read -p "Enter selection [0-3] > "

if [[ $REPLY =~ ^[0-3]$ ]]; then
      if [[ $REPLY == 0 ]]; then
             echo "Program terminated."
             exit
      fi
      if [[ $REPLY == 1 ]]; then
             echo "Hostname: $HOSTNAME"
             uptime
             exit
      fi
      if [[ $REPLY == 2 ]]; then
             df -h
             exit
      fi
      if [[ $REPLY == 3 ]]; then
             if [[ $(id -u) -eq 0 ]]; then
                    echo "Home Space Utilization (All Users)"
                    du -sh /home/*
             else
                    echo "Home Space Utilization ($USER)"
                    du -sh $HOME
                fi
                exit
       fi
else
       echo "Invalid entry." >&2
       exit 1
```

```
fi
```

يُقسَّم السكربت السابق منطقيًّا إلى قسمَين: القسم الأول يُظهِر القائمة ويقبل المدخلات من المستخدم. والقسم الثاني يتحقق من المدخلات وينفذ الأمر المُحدَّد. لاحظ استخدام الأمر exit في السكربت، حيث استخدمناه لمنع السكربت من تنفيذ الأكواد غير الضرورية بعد تنفيذ ما قام المستخدم باختياره. وجود أكثر من عبارة exit في مختلف المواضع في السكربت هو أمرٌ غير مستحب (لأنه يجعل فهم التسلسل المنطقي للبرنامج صعبًا)، لكنه يعمل عملًا جيدًا في هذا السكربت.

الخلاصة

لقد خطونا أولى خطواتنا نحو التفاعليّة في هذا الفصل؛ حيث سمحنا للمستخدمين بإدخال البيانات إلى برنامجنا باستخدام لوحة المفاتيح. أصبح بإمكاننا إنشاء برامج مفيدة وتفاعليّة باستخدام هذه الآلية. من الممكن كتابة العديد من البرامج المفيدة، كالبرامج الحسابية المتخصصة وواجهات مُبَسَّطة لأوامر معقدة. سنبني -في الفصل القادم- على البرنامج الذي يحتوي القائمة لتحسينه وتطويره.

أضف إلى معلوماتك

من المهم دراسة البرامج الموجودة في هذا الفصل جيدًا، وفهم طريقة بنائها فهمًا كاملًا؛ لأن البرامج التالية ستزداد تعقيدًا. كتمرينٍ على هذا الفصل، أعد كتابة الأمثلة الموجودة فيه مستخدمًا الأمر test بدلًا من الأمر المركب [[ ]]. تلميحة: استخدم الأمر grep للتحقق من التعابير النظامية ثم تحقق من حالة خروجه.

الفصل التاسع والعشرون:

بُنى التحكم: التكرار باستخدام while/until

لقد طورّنا في الفصل السابق البرنامج الذي يعرض مختلف المعلومات عن النظام، حيث أصبح يحتوي على قائمة اختيارات. يعمل ذاك البرنامج لكنه يعاني من مشكلة في قابلية الاستخدام. إنه ينفذ خيارًا واحدًا فقط ومن ثم ينتهي! وحتى أسوأ من ذلك، فعند تحديد خيار غير موجود في القائمة فسينتهي تنفيذ البرنامج مع إظهار رسالة خطأ؛ دون إعطاء المستخدم فرصةً للتجربة مرّةً ثانيةً. سيكون من الأفضل تطوير البرنامج لكي يُعيد إظهار القائمة بعد تنفيذ أحد خياراتها إلى أن يختار المستخدم الخروج من البرنامج.

سنلقي في هذا الفصل نظرة على أحد جوانب البرمجة الذي يسمى "التكرار" (looping)، حيث يسمح بتكرار أحد أجزاء البرنامج. توفر الصدفة ثلاث بُنى تُستخدم للتكرار؛ سنلقي نظرة على اثنتين منها في هذا الفصل، وسنؤجل الثالثة إلى فصلٍ قادم.

التكرار

الحياة العملية مليئة بالأعمال المتكررة. الذهاب إلى العمل كل يوم، وتناول وجبات الطعام، وتقسيم الجزر هي أمثلة عن المهام التي تحتوي على سلسلة خطوات مكررة. لنفترض أننا نريد كتابة خوارزمية تقطيع الجزر. يمكن التعبير عن تلك الخوارزمية كالآتي:

1. جَهِّز لوح التقطيع.
2. امسك السكين.
3. ضع الجزرة على لوح التقطيع.
4. ارفع السكين.
5. حرّك الجزرة إلى الأمام.
6. اقطع الجزرة.
7. قم بإنهاء المهمة عند إكمال تقطيع الجزرة؛ عدا ذلك، نفذ الخطوة 4.

تُشكّل الخطوات من 4 إلى 7 "حلقة تكرار". ستُنفَّذ الخطوات الموجودة داخل الحلقة حتى يتحقق الشرط "إكمال تقطيع الجزرة".

while

يمكن للصدفة bash التعبير عن تلك الفكرة باستخدام حلقة التكرار while. لنفترض أننا نريد إظهار خمسة أرقام بالتسلسل من الواحد إلى الخمسة، فسيكون سكربت bash كالآتي:

```
#!/bin/bash

# while-count: display a series of numbers

count=1

while [ $count -le 5 ]; do
        echo $count
        count=$((count + 1))
done
echo "Finished."
```

سيُظهِر السكربت الناتج الآتي:

```
[me@linuxbox ~]$ while-count
1
2
3
4
5
Finished.
```

الشكل العام لحلقة التكرار while هو:

`while commands; do commands; done`

ستتحقق حلقة while من حالة الخروج لقائمة الأوامر كما في if. ستنفَّذ الأوامر داخل الحلقة طالما كانت حالة الخروج تساوي 0.

أنشأنا، في السكربت السابق، المتغير count وأسندنا القيمة 1 له. ستتحقق while من قيمة حالة خروج الأمر test. وستُنفَّذ الأوامر داخل الحلقة لطالما كانت حالة الخروج للأمر test تساوي الصفر. سيعاد تنفيذ الأمر test بعد الانتهاء من تنفيذ الأوامر داخل الحلقة. ستزداد قيمة المتغير count إلى 6 بعد ستة تكرارات للحلقة. عندها لن يُعيد الأمر test حالة خروج تساوي 0، وستنتهي الحلقة. سيكمل البرنامج بعدها تنفيذ

الأوامر التي تلي الحلقة.

يمكننا استخدام while لتحسين برنامج read-menu الذي أنشأناه في الفصل السابق:

```
#!/bin/bash

# while-menu: a menu driven system information program

DELAY=3 # Number of seconds to display results

while [[ $REPLY != 0 ]]; do
	clear
	cat <<- _EOF_
		Please Select:
		1.  Display System Information
		2.  Display Disk Space
		3.  Display Home Space Utilization
		0.  Quit
	_EOF_
	read -p "Enter selection [0-3] > "
	if [[ $REPLY =~ ^[0-3]$ ]]; then
		if [[ $REPLY == 1 ]]; then
			echo "Hostname: $HOSTNAME"
			uptime
			sleep $DELAY
		fi
		if [[ $REPLY == 2 ]]; then
			df -h
			sleep $DELAY
		fi
		if [[ $REPLY == 3 ]]; then
			if [[ $(id -u) -eq 0 ]]; then
			      echo "Home Space Utilization (All Users)"
			      du -sh /home/*
			else
			      echo "Home Space Utilization ($USER)"
			      du -sh $HOME
```

```
                        fi
                        sleep $DELAY
                fi
        else
                echo "Invalid entry."
                sleep $DELAY
        fi
done
echo "Program terminated."
```

استطعنا جعل البرنامج يعيد إظهار القائمة بعد كل اختيار بتضمين القائمة في حلقة while، سيستمر تنفيذ الحلقة طالما كانت قيمة المتغير REPLY لا تساوي "0"؛ حيث تظهر قائمة الاختيارات مرّةً أُخرى ويُعطى المستخدم إمكانية تحديد خيارٍ آخر. سننفذ الأمر sleep بعد نهاية كل أمر في الحلقة كي يوقف البرنامج عن العمل لفترة وجيزة من الزمن كي يستطيع المستخدم قراءة الناتج قبل مسح محتويات الشاشة (الأمر clear) وإظهار القائمة مرّة أُخرى.

سينتهي تنفيذ الحلقة عندما تكون قيمة المتغير REPLY تساوي "0" (أي قد حُدِّدَ الخيار "quit") وستُنفَّذ الأوامر التي تلي الكلمة done.

الخروج من الحلقة

توفر bash أمرَين مضمنين فيها يمكن استخدامهما للتحكم في سَير البرنامج داخل الحلقة: الأمر break، الذي يُنهي الحلقة مباشرةً، وسيكمل البرنامج تنفيذ الأوامر التي تلي الحلقة؛ والأمر continue الذي يؤدي إلى تخطي ما تبقى من الحلقة، وبعد ذلك ستكمل الحلقة تنفيذها. هذه نسخة من برنامج while-menu تستخدم break و continue للتحكم في الحلقة:

```
#!/bin/bash

# while-menu2: a menu driven system information program

DELAY=3 # Number of seconds to display results

while true; do
        clear
        cat <<- _EOF_
```

```
                Please Select:
                1. Display System Information
                2. Display Disk Space
                3. Display Home Space Utilization
                0. Quit
        _EOF_
        read -p "Enter selection [0-3] > "
        if [[ $REPLY =~ ^[0-3]$ ]]; then
                if [[ $REPLY == 1 ]]; then
                      echo "Hostname: $HOSTNAME"
                      uptime
                      sleep $DELAY
                      continue
                fi
                if [[ $REPLY == 2 ]]; then
                      df -h
                      sleep $DELAY
                      continue
                fi
                if [[ $REPLY == 3 ]]; then
                      if [[ $(id -u) -eq 0 ]]; then
                             echo "Home Space Utilization (All Users)"
                             du -sh /home/*
                      else
                              echo "Home Space Utilization ($USER)"
                              du -sh $HOME
                       fi
                       sleep $DELAY
                       continue
               fi
               if [[ $REPLY == 0 ]]; then
                       break
               fi
        else
               echo "Invalid entry."
               sleep $DELAY
```

```
        fi
done
echo "Program terminated."
```

في هذه النسخة من السكربت، أنشأنا حلقة تكرار لا نهائيـة (الحلقة التي لا تحتـوي على شرط لإنهائها) وذلك باستخدام الأمر true لكي يوفر حالة الخروج لحلقة while. ولما كانت حالة الخروج للأمر true هي دائمًا 0، فلن يتم إنهاء الحلقة while. هذه التقنية مُستخدمة بكثرة في السكربتات. ولأن الحلقة لا نهائية، فإن على البرنامـج أن يـوفر آليـة للخـروج مـن الحلقـة عنـد حـدوث شـروط معينـة. فـي السـكربت السـابق، أُسـتخدم الأمـر break للخـروج مـن الحلقـة عنـد تحديـد الخيـار "0". يُسـتخدم الأمـر continue فـي نهايـة بـاقي الخيـارات لتخطي باقي الأسطر البرمجيـة التي لا حاجة لها. على سبيل المثال، إذا حُدِّدَ الخيـار "1"، فليـس هنالك سببٌ لتنفيذ باقي الاختبارات.

until

الأمر until شبيه للغاية بالأمر while، باستثناء أنه عوضًا عن الخروج من الحلقة عندما تكون حالة الخـروج للأوامر المُنفَّذة لا تساوي الصفر، فإنه يفعل العكس تمامًا. تستكمل حلقة التكرار until تنفيـذها إلى أن تكون حالة الخـروج للأوامر المُنفَّذة تسـاوي 0. في سـكربت while-count السـابق، كررنا الحلقـة إلى أن أصبحت قيمة المتغير count أصغر أو تساوي 5. يمكننا الحصول على نفس النتيجة باستخدام حلقة until:

```
#!/bin/bash

# until-count: display a series of numbers

count=1

until [ $count -gt 5 ]; do
        echo $count
        count=$((count + 1))
done
echo "Finished."
```

ستنتهي حلقة until في الوقت المناسب بتغيير تعبير الاختبار إلى $count -gt 5. يتعلق الاختيار ما بين استخدام while أو until بشكلٍ أساسي بأيّة حلقة تسمح بكتابة تعبير الاختبار بأبسط صيغة ممكنة.

قراءة الملفات باستخدام حلقات التكرار

يمكن لحلقتي التكرار while و until قبول المدخلات من مجرى الدخل القياسي. وهذا ما يسمح بأن تُعالَج الملفات باستخدام تلك الحلقتَين. في المثال الآتي، سنُظهر محتويات ملف distors.txt الذي استخدمناه في فصولٍ سابقة:

```
#!/bin/bash

# while-read: read lines from a file

while read distro version release; do
        printf "Distro: %s\tVersion: %s\tReleased: %s\n" \
                $distro \
                $version \
                $release
done < distros.txt
```

لإعادة توجيه الملف إلى الحلقة، وضعنا معامل إعادة التوجيه بعد عبارة done. تستخدم الحلقة الأمر read لقراءة الحقول من الملف. سينتهي الأمر read بعد قراءة كل سطر بحالة خروج تساوي 0 إلى أن يصل إلى نهاية الملف (EOF). حيث سيتم إعادة حالة خروج لا تساوي الصفر مما يؤدي إلى إنهاء الحلقة. من الممكن أيضًا أن تُستخدم الأنابيب مع حلقة التكرار:

```
#!/bin/bash

# while-read2: read lines from a file

sort -k 1,1 -k 2n distros.txt | while read distro version release; do

        printf "Distro: %s\tVersion: %s\tReleased: %s\n" \
                $distro \
                $version \
                $release
done
```

أخذنا مخرجات الأمر sort ومررناها إلى الحلقة. لكن من المهم جدًا تذكر أنه ولما كانت الأنابيب تُنَفَّذ في صدفة فرعية، فإن أي متغير تم إنشاؤه أو إسناد قيمة إليه "سيُدمَّر" عند انتهاء تنفيذ الحلقة.

الخلاصة

بعد تعرّفنا على الحلقات، ومن خبرتنا السابقة في التفرعات، وتعلمنا للاختبارات، فإننا غطينا أشهر الآليات المستخدمة للتحكم في سَير البرامج. لكن الصدفة bash تحتوي المزيد في جُعبَتِها، لكن ما تبقى هو مبنيٌ على هذه الأمور الأساسية.

الفصل الثلاثون:

استكشاف الأخطاء وإصلاحها

حان الوقت الآن لإلقاء نظرة على الذي سيحصل عندما تحدث الأخطاء ويقوم البرنامج بأفعال لا نريدها أو لا نتوقعها. سنناقش في هذا الفصل بعض أنواع الأخطاء التي تحدث في السكربتات وسنشرح بعض الآليات المُستخدمة لِتَتَبُّع وإزالة الأخطاء.

الأخطاء البنيوية

فئة من الفئات العامة للأخطاء هي الأخطاء البنيوية؛ الخطأ البنيوي يكون عبارة عن خطأ مطبعي في أسماء أو أشكال بعض عناصر ومكونات السكربت. تؤدي هذه الأخطاء في أغلب الحالات إلى رفض تشغيل السكربت من قِبل الصدفة.

سنستخدم في الفقرات التالية السكربت الآتي لشرح مختلف أنواع الأخطاء:

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=1

if [ $number = 1 ]; then
	echo "Number is equal to 1."
else
	echo "Number is not equal to 1."
fi
```

سيُنفَّذ السكربت السابق بنجاح ويعطي الخرج الآتي:

```
[me@linuxbox ~]$ trouble
Number is equal to 1.
```

علامة اقتباس ناقصة

إذا عدّلنا في السكربت الأصلي وأزلنا علامة الاقتباس من الوسيط الممرر إلى أول أمر echo في السكربت:

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=1

if [ $number = 1 ]; then
        echo "Number is equal to 1.
else
        echo "Number is not equal to 1."
fi
```

فلاحظ ماذا سيحدث:

```
[me@linuxbox ~]$ trouble
/home/me/bin/trouble: line 10: unexpected EOF while looking for  match-
ing `"'
/home/me/bin/trouble: line 13: syntax error: unexpected end of file
```

لقد وَلَّد السكربت خطأين اثنين. لاحظ أن أرقام الأسطر التي بُلِّغَ بوجود خطأٍ فيها (وبشكل مثير للاستغراب) لا تتضمن رقم السطر الذي يحتوي على علامة اقتباس ناقصة، بل بعد ذاك السطر بكثير. يمكننا معرفة سبب ذلك إذا تتبعنا ماذا حصل للبرنامج بعد علامة الاقتباس الناقصة. ستستمر الصدفة في البحث عن علامة إغلاق الاقتباس حتى تجدها، تحديدًا بعد أمر echo الثاني مباشرةً. ستصبح الأمور بعد ذلك مربكة جدًا بالنسبة للصدفة؛ حيث حدث خطأ في تركيب وبنية الدالة الشرطية `if`، لأن جزءًا منها أصبح داخل العبارة المُقتبَسة! قد يصبح العثور على هذا النوع من الأخطاء في السكربتات الطويلة أمرًا صعبًا وشاقًّا للغاية؛ لذا، سيساعد استخدام محرر يوفر ميزة تلوين الأكواد مساعدةً كبيرةً على كشف هذه الأخطاء. إذا كانت النسخة الكاملة من `vim` مُثبَّتة على جهازك، فتستطيع تفعيل تلوين الأكواد بإدخال الأمر الآتي:

```
:syntax on
```

أجزاء ناقصة من البُنى

خطأ شائع آخر هو نسيان إكمال أمر بنيوي، كالدالة الشرطية if أو حلقة التكرار while. لنلاحظ ماذا سيحصل عند إزالة الفاصلة المنقوطة بعد الأمر test في if:

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=1

if [ $number = 1 ] then
        echo "Number is equal to 1."
else
        echo "Number is not equal to 1."
fi
```

فتكون النتيجة كالآتي:

```
[me@linuxbox ~]$ trouble
/home/me/bin/trouble: line 9: syntax error near unexpected token `else'
/home/me/bin/trouble: line 9: `else'
```

مرة أُخرى، تشير رسالة الخطأ إلى سطر موجود بعد السطر الذي حدثت فيه المشكلة. المشكلة التي حدثت مثيرة للاهتمام. كما نتذكر سابقًا، يقبل if قائمة من الأوامر ويتحقق من حالة الخروج لآخر أمرٍ منها. أردنا (في السكربت السابق) أن تحتوي تلك القائمة على أمرٍ واحدٍ فقط: "[" الذي يشير إلى الأمر test. الأمر [يقبل الأوامر التي تليه كوسائط؛ في هذه الحالة هي أربعة وسائط: $number، و =، و 1، و]. وعند إزالة الفاصلة المنقوطة، أُضيفت الكلمة then إلى قائمة الوسائط (يُسمَح قواعديًا بذلك). يُسمَح أيضًا بالأمر echo، حيث سيُفسَّر على أنه أمر من قائمة الأوامر التي سيتحقق if من قيمة حالة خروجها. ومن ثم تظهر الكلمة else، وهنا تظهر المشكلة، حيث أن else ليست في مكانها الصحيح، ولأن الصدفة تُعرِّف else على أنها "كلمة محجوزة" (reserved word – كلمة ذات معنى خاص بالنسبة إلى الصدفة) وليست اسم أمر، فستطبع رسالة الخطأ.

الأخطاء غير المتوقعة

من الممكن حدوث أخطاء محددة بشكل غير متوقع. أي أن السكربت سيُنفَّذ تنفيذًا سليمًا في بعض الأحيان،

ويفشل في أحيانٍ أُخرى. إذا عدّلنا السكربت السابق وأسندنا قيمةً فارغةً إلى المتغير number:

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=

if [ $number = 1 ]; then
	echo "Number is equal to 1."
else
	echo "Number is not equal to 1."
fi
```

يؤدي تشغيل هذا السكربت إلى إظهار النتائج الآتية:

```
[me@linuxbox ~]$ trouble
/home/me/bin/trouble: line 7: [: =: unary operator expected
 Number is not equal to 1.
```

حصلنا على رسالة خطأ مبهمة، يتبعها مخرجات أمر echo الثاني. المشكلة تحدث عند توسعة المتغير number في الأمر test:

```
[ $number = 1 ]
```

ولما كان المتغير number فارغًا، فإن نتيجة التوسعة هي:

```
[  = 1 ]
```

وهي عملية غير صحيحة، مما يؤدي إلى توليد رسالة الخطأ. المعامل = هو معامل ثنائي (يتطلب قيمة واحدة على كل جهة)، لكن القيمة الأولى غير موجودة، لذا، فإن الأمر test يتوقع استخدام معامل أحادي (z- على سبيل المثال). وبسبب فشل تنفيذ الأمر test (بسبب حدوث الخطأ)، فإن الدالة الشرطية if تستقبل حالة خروج لا تساوي الصفر، مما يؤدي إلى تنفيذ أمر echo الثاني.

يمكن أن تُصَحَّح تلك المشكلة بإحاطة المتغير بعلامتَيّ اقتباس، كما يلي:

```
[ "$number" = 1 ]
```

فعندما تتم التوسعة؛ سيكون الناتج هو:

```
[ "" = 1 ]
```

مما يحقق العدد الصحيح من الوسائط. بالإضافة إلى السلاسل النصية الفارغة، يجب استخدام الاقتباس في القيم التي ستتوسع إلى سلسلة نصية تحتوي على أكثر من كلمة، كأسماء الملفات التي تحتوي على فراغات.

الأخطاء المنطقية

على النقيض من الأخطاء البنيوية، لا تمنع الأخطاء المنطقية تنفيذ السكربت. سيعمل السكربت لكنه لن يعطي النتائج المطلوبة بسبب مشكلة في البنية المنطقية له. لا يمكن إحصاء جميع الأخطاء المنطقية، لكننا سنذكر أشهرها:

التعابير الشرطية الخاطئة. من السهل كتابة تعبير `if/then/else` يحتوي على خطأ منطقي. في بعض الأوقات يكون الخطأ في أن التعبير مقلوب (أي ما يجب تنفيذه في `if` يُنفَّذ في `else`)، أو أنه غير كامل.

1. **الأخطاء في التكرارات.** عند كتابة حلقات التكرار التي تقوم بالعدّ، من الممكن أن لا تصل الحلقة إلى الرقم المطلوب أو أن تزيد عن العدد المطلوب؛ وذلك بسبب وجود خطأ منطقي في شرط التكرار.
2. **الأخطاء غير المتوقعة.** أغلب الأخطاء المنطقية تكون نتيجةً لحدوث حالات لم يحسب المبرمج حسابها. كاستخدام اسم ملف يحتوي على فراغات، مما يؤدي إلى اعتباره وسائط للأمر المُنفَّذ بدل أن يكون اسم ملف.

اتباع طريقة وقائية في البرمجة

من المهم التحقق من حالات الخروج للأوامر المُستخدمة عند كتابة السكربتات. لنفرض هذا المثال المبني على قصة حقيقية. كتب مدير أنظمة غير محظوظ سكربتًا لصيانة خادم مهم. حوى ذاك السكربت على السطرين الآتيين:

```
cd $dir_name
rm *
```

لا يوجد شيء خاطئ في هذين السطرين ما دامت قيمة المتغير `dir_name` تمثل مسار مجلد موجود في النظام. لكن ماذا سيحصل إن لم يكن ذاك المجلد موجودًا؟ سيفشل تنفيذ الأمر `cd` في تلك الحالة وسيتم الانتقال إلى السطر الآتي الذي يمسح جميع الملفات في مجلد العمل الحالي. وهنا تقع الكارثة! دمّر مدير النظام البائس جزءًا مهمًا من الخادم بسبب سوء تصميمه للسكربت.

لنلقِ نظرة على بعض الطرق التي يمكنها تحسين تصميم السكربت السابق. من الحكمة جعل تنفيذ الأمر rm مرتبطًا بنجاح تنفيذ الأمر cd:

```
cd $dir_name && rm *
```

في هذه الحالة، إذا فشل تنفيذ الأمر cd فلن يُنفَّذ الأمر rm. وهذا أفضل بكثير من التصميم السابق لكنه يفتح المجال أمام إمكانية عدم تعيين المتغير dir_name أو أن تكون قيمته فارغة، مما يؤدي إلى حذف محتويات مجلد المنزل للمستخدم المُنفِّذ للسكربت. يمكن تخطي هذه الإشكالية بالتحقق فيما إذا كان المتغير dir_name يحتوي على مسار مجلد موجود مسبقًا:

```
[[ -d $dir_name ]] && cd $dir_name && rm *
```

قد يكون من الجيد أن ننهي البرنامج بعد إرسال رسالة خطأ عند مواجهة إحدى الإشكاليات السابقة:

```
if [[ -d $dir_name ]]; then
        if cd $dir_name; then
                rm *
        else
                echo "cannot cd to '$dir_name'" >&2
                exit 1
        fi
else
        echo "no such directory: '$dir_name'" >&2
        exit 1
fi
```

تحققنا من اسم المجلد (لكي نعرف إن كان المجلد موجودًا) ونجاح تنفيذ الأمر cd. إذا فشل أحدهما، فستُطبَع رسالة خطأ إلى مجرى الخطأ القياسي وسينتهي السكربت بحالة خروج لا تساوي الصفر للإشارة إلى فشل تنفيذه.

التحقق من المدخلات

سمة عامة من سمات الأسلوب البرمجي الجيد هي إمكانية معالجة البرنامج لأيّة مدخلات يستقبلها (في حال كان البرنامج يستقبل أيّة مدخلات). هذا يعني أنه يتأكد من أن المدخلات الصحيحة فقط ستُقبَل لكي تُعالَج. لقد شاهدنا مثالًا على ذلك في الفصل الماضي عندما ناقشنا الأمر read. كان أحد السكربتات يحتوي على الاختبار الآتي للتحقق من صحة اختيار المستخدم لأحد خيارات القائمة:

```
[[ $REPLY =~ ^[0-3]$ ]]
```

يُعيد الاختبار السابق حالة خروج تساوي الصفر إذا كانت السلسلة المُدخلَة من قِبل المستخدم هي عددٌ موجود في المجال من 0 إلى 3. ولن يُقبَل أيّ شيء آخر. قد تكون كتابة مثل هذه الاختبارات صعبة في بعض الأحيان، لكن هذا الجهد ضروري لإنشاء سكربتات ذات جودة عالية.

العلاقة الطردية ما بين التصميم والوقت

ذكر أستاذ جامعي حكيم أن جودة تصميم مشروعٍ ما ترتبط بمقدار الوقت المعطى للمصمم. إذا أُعطيتَ خمس دقائق لتصميم جهاز "يقتل الذباب"، فسينتهي بك المطاف بتصميم "مِلَطَشة ذباب". أما إذا أُعطيت خمسة أشهر، فستصمم منظومة دفاع جوي ضد الذباب موجهة بالليزر!

نفس المبدأ ينطبق على البرمجة. في بعض الأحيان، تُكتَب السكربتات التي تُستخدم لمرّة واحدة فقط من قِبل مبرمجها بسرعة دون كتابة الكثير من التعليقات أو القيام بالعديد من التحققات. لكن على الكفة الأُخرى، توجد السكربتات التي تحتاج إلى التصميم الحذر الموجهة للاستخدام "الإنتاجي"، أي السكربتات التي تُستخدم للقيام بمهمة مِحورية أو التي تُستخدم من قِبل عدّة مُستخدمين.

التجربة

تجريب البرامج هو خطوة مهمة في عملية تطوير البرمجيات، بما فيها السكربتات. هنالك مقولة في عالم المصادر المفتوحة: "أصدر برنامجك مبكرًا وبين الحين والآخر". تعكس تلك العبارة أهمية تجربة البرامج. فعند "إصدار البرنامج مبكرًا وبين الحين والآخر"، فسيكون البرنامج عرضةً للاستخدام والتجربة. من الواضح أنه كلما عُثِرَ على العلل في وقت مبكر من مرحلة التطوير، كلما سَهُل العثور على مسببها ثم إصلاحها. شاهدنا في نقاشنا السابق بعض التقنيات التي قد تُستخدم للتحقق من سير البرنامج من بداية تطويره، تلك التقنيات مفيدة جدًا للتحقق من تقدم تطوير البرنامج.

لنعد إلى سكربت حذف الملفات السابق ولننظر كيف يمكن تطويره لتسهيل عملية التجربة. تجربة الكود الأصلي للبرنامج هو أمرٌ خطير، لأنه يحذف الملفات، لذا سنحتاج إلى تعديل الكود لكي يصبح "آمنًا":

```
if [[ -d $dir_name ]]; then
        if cd $dir_name; then
                echo rm * # TESTING
        else
                echo "cannot cd to '$dir_name'" >&2
```

```
                exit 1
        fi
else
        echo "no such directory: '$dir_name'" >&2
        exit 1
fi
exit # TESTING
```

لا نحتاج إلى إضافة رسائل خطأ لأنها موجودة بالفعل. أهم تعديل قمنا به هو إضافة الأمر echo قبل الأمر rm لإظهار الأمر المُنفَّذ مع قائمة الوسائط عوضًا عن تنفيذه مباشرةً. يسمح لنا هذا التعديل بتجربة البرنامج بأمان. أضفنا الأمر exit في آخر السكربت كي ننهي السكربت عند هذا الحد ونمنعه من تنفيذ أيّة أجزاء أُخرى؛ ضرورة إضافة هذا الأمر تختلف من سكربتٍ إلى آخر.

أضفنا أيضًا بعض التعليقات لكي نُعَلِّم التعديلات التي أجريناها لغرض التجربة. تفيد هذه التعليقات بالعثور على التعديلات بعد انتهاء التجارب لإزالتها.

حالات التجربة

من الضروري أن نطور بضع "حالات تجربة" (test cases) لتجربة البرنامج تجربةً وافيةً؛ وذلك بالاختيار الحذر للمدخلات أو شروط عمل البرنامج لكي نَحصُر جميع الحالات. الكود السابق (الذي هو بسيط للغاية) يعمل تحت الشروط الثلاثة الآتية:

1. المتغير dir_name يحتوي على مسار مجلد موجود مسبقًا.
2. المتغير dir_name يحتوي على مسار مجلد لكنه غير موجود.
3. المتغير dir_name لا يحتوي على أيّة قيمة.

بالقيام بالاختبارات وفق الشروط الثالثة السابقة، فإننا سنغطي جميع الاحتمالات. وكما في التصميم، فإن التجربة تتناسب طردًا مع الزمن؛ ولا حاجة لاختبار جميع ميزات السكربت وخصائصه. من الجيد تحديد الأمور المهمة التي يجب تجربتها وخصوصًا تلك التي تُحدِث أضرارًا عند حدوث علّة أو مشكلة فيها.

التنقيح

إذا كشفت مرحلة الاختبار عن وجود مشكلة في السكربت، فالخطوة التالية هي التنقيح (debugging). "المشكلة" تعني عادةً أن السكربت لا يقوم، بشكلٍ أو بآخر، بما يتوقعه المبرمج. إذا كان كذلك، فعلينا أن نحدد بدقة ما الذي يفعله السكربت ولماذا. قد يحتاج اكتشاف العلل إلى تحقيقات كثيرة.

سيساعدك التصميم الجيد للبرامج في تنقيحها لأنك تتبع الطريقة الوقائية في البرمجة؛ لأنها تُوفِّر آلية لإظهار معلومات للمستخدم عند حدوث أمر غير متوقع. لكن في بعض الأحيان، تكون المشكلة غريبة جدًا أو غير متوقعة؛ لذا، سنحتاج إلى استخدام تقنيات أُخرى لكشفها.

عزل المنطقة المصابة

من المفيد أحيانًا عزل المنطقة من السكربت التي تحدث فيها المشاكل؛ وذلك في بعض السكربتات وخصوصًا الطويلة منها. لن تكتشف هذه الطريقة الخطأ دائمًا؛ لكن يعطينا العزل فكرةً عن السبب الحقيقي للمشكلة. إحدى الطرق المُستخدمة لعزل الأكواد هي إدراج بعض أجزاء السكربت في تعليق. على سبيل المثال، يمكن تعديل سكربت حذف الملفات لتحديد فيما إذا كان القسم المحذوف متعلقًا بخطأٍ ما:

```
if [[ -d $dir_name ]]; then
	if cd $dir_name; then
		rm *
	else
		echo "cannot cd to '$dir_name'" >&2
		exit 1
	fi
# else
#	echo "no such directory: '$dir_name'" >&2
#	exit 1
fi
```

لقد منعنا قسمًا من السكربت من التنفيذ بإضافة إشارة التعليق إلى بداية كل سطر من ذاك القسم. علينا الآن تجربة السكربت المُعدَّل لنرَ إذا أثّر حذف القسم السابق على سلوك العلّة.

التَتّبع

تسبب العِلل عادةً أحداثًا غير متوقعة داخل السكربت. هذا يعني أن أجزاءً من السكربت لا تُنفَّذ أبدًا، أو أن تُنَفَّذ بترتيب أو توقيت غير صحيح. نستخدم تقنية تسمى التَتَبّع (tracing) لكي نعرف خط سير السكربت الفعلي.

إحدى طرق التَتَبّع تتضمّن وضع رسائل تحتوي على معلومات في السكربت التي تظهر مكان التنفيذ. بإمكاننا إضافة الرسائل إلى السكربت السابق:

```
echo "preparing to delete files" >&2
```

```
if [[ -d $dir_name ]]; then
      if cd $dir_name; then
echo "deleting files" >&2
              rm *
      else
              echo "cannot cd to '$dir_name'" >&2
              exit 1
      fi
else
      echo "no such directory: '$dir_name'" >&2
      exit 1
fi
echo "file deletion complete" >&2
```

أرسلنا الرسائل إلى مجرى الخطأ القياسي لعزلهم عن الناتج العادي. ولم نحاذِ الأسطر التي تحتوي على الرسائل، لكي يصبح العثور عليها سهلًا عندما نريد حذفها.

أصبح بإمكاننا أن نشاهد أن الملف قد حُذِف عند تنفيذ السكربت المُعدَّل:

```
[me@linuxbox ~]$ deletion-script
preparing to delete files
deleting files
file deletion complete
[me@linuxbox ~]$
```

توفر bash طريقة للتتبع، وذلك باستخدام الخيار x-، والأمر set مع الخيار x-. سنُفَعِّل التتبع في سكربت trouble السابق بإضافة الخيار x- إلى أول سطر:

```
#!/bin/bash -x

# trouble: script to demonstrate common errors

number=1

if [ $number = 1 ]; then
      echo "Number is equal to 1."
else
```

```
      echo "Number is not equal to 1."
fi
```

ستكون النتائج كالآتي عند تنفيذ السكربت:

```
[me@linuxbox ~]$ trouble
+ number=1
+ '[' 1 = 1 ']'
+ echo 'Number is equal to 1.'
Number is equal to 1.
```

سنشاهد الأوامر التي تُنفَّذ مع التوسعات التي تحويها عندما نُفَعِّل التتبع. تُستخدم إشارة الزائد الموجودة في أول السطر للتفريق ما بين المخرجات العادية ومخرجات التتبع. إشارة الزائد هي الإشارة الافتراضية للتتبع؛ وهي موجودة في متغير الصدفة PS4 (prompt string 4). يمكن تعديل محتويات هذا المتغير لكي نجعل المِحث أكثر فائدةً. عدّلنا -في المثال الآتي- محتويات ذاك المتغير لإضافة رقم السطر في السكربت الذي يُنَفَّذ عليه التتبع. لاحظ أن علامتَي الاقتباس المفردتَين ضروريتَين لكي لا تُنَفَّذ التوسعة إلى أن يُستخدَم المِحث:

```
[me@linuxbox ~]$ export PS4='$LINENO + '
[me@linuxbox ~]$ trouble
5 + number=1
7 + '[' 1 = 1 ']'
8 + echo 'Number is equal to 1.'
Number is equal to 1.
```

لتتبع جزء مُعيَّن من سكربت، بدلًا من كامل السكربت، نستخدم الأمر set مع الخيار "x-":

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=1

set -x # Turn on tracing
if [ $number = 1 ]; then
      echo "Number is equal to 1."
else
      echo "Number is not equal to 1."
```

```
fi
set +x # Turn off tracing
```

استخدمنا الأمر set مع الخيار x- لتفعيل التتبع، والخيار x+ لتعطيل التتبع. يمكن استخدام هذه التقنية لمعاينة أجزاء مختلفة من سكربت يحتوي على العديد من المشاكل.

معاينة القيم أثناء التنفيذ

من المفيد عادةً، إلى جانب التتبع، أن تُظهَر قيم المتغيرات لكي نعرف القيم المخزّنة فيها أثناء التنفيذ. يمكن استخدام الأمر echo لتنفيذ هذه المهمة:

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=1

echo "number=$number" # DEBUG
set -x # Turn on tracing
if [ $number = 1 ]; then
       echo "Number is equal to 1."
else
       echo "Number is not equal to 1."
fi
set +x # Turn off tracing
```

في المثال السابق المتواضع، أظهرنا بكل بساطة قيمة المتغير number وعلّمنا السطر المضاف بتعليق كي نستطيع تميزه لاحقًا لكي نحذفه. هذه التقنية مفيدة خصوصًا عند مراقبة سلوك حلقات التكرار والتعابير الرياضية داخل السكربتات.

الخلاصة

لقد ألقينا نظرة، في هذا الفصل، على العديد من المشاكل التي قد تفاجئنا أثناء تطوير السكربتات. ما زال هنالك الكثير! ستُمكّنك التقنيات التي ناقشناها هنا من اكتشاف أغلب أنواع العِلَل الشائعة. التنقيح هو فن رائع يمكنك تطوير قدراتك فيه؛ وذلك في تجنب العِلَل (بالتجربة المتواصلة أثناء التطوير) والعثور على العِلَل (باستخدام التتبع).

الفصل الحادي والثلاثون:

بُنى التحكم: التفرع باستخدام case

سنكمل في هذا الفصل شرحنا لبُنى التحكم. في الفصل 28، أنشأنا بعض القوائم البسيطة وكتبنا البنية المنطقية للبرنامج كي يستجيب إلى اختيارات المستخدم. قمنا بذلك باستخدام سلسلة من أوامر if للتحقق من أن أحد الخيارات قد أُختير. توفر العديد من لغات البرمجة، (بما فيها الشِل) بُنية تحكم لاختبار العديد من الحالات.

case

الأمر البنيوي الموجود في bash الذي يُستخدم للاختيار المتعدد يسمى case. الصياغة العامة هي:

```
case word in
        [pattern [| pattern]...) commands ;;]...
esac
```

إذا ألقينا نظرةً على برنامج read-menu من الفصل 28، سنشاهد البنية المنطقية التي يستخدمها البرنامج لتنفيذ اختيار المستخدم:

```
#!/bin/bash

# read-menu: a menu driven system information program

clear
echo "
Please Select:

1. Display System Information
2. Display Disk Space
3. Display Home Space Utilization
0. Quit
"
read -p "Enter selection [0-3] > "
```

```
if [[ $REPLY =~ ^[0-3]$ ]]; then
     if [[ $REPLY == 0 ]]; then
            echo "Program terminated."
            exit
     fi
     if [[ $REPLY == 1 ]]; then
           echo "Hostname: $HOSTNAME"
           uptime
           exit
     fi
     if [[ $REPLY == 2 ]]; then
           df -h
           exit
     fi
     if [[ $REPLY == 3 ]]; then
           if [[ $(id -u) -eq 0 ]]; then
                  echo "Home Space Utilization (All Users)"
                  du -sh /home/*
           else
                  echo "Home Space Utilization ($USER)"
                  du -sh $HOME
           fi
           exit
     fi
else
        echo "Invalid entry." >&2
        exit 1
fi
```

يمكن استبدال البنية المنطقية للمثال السابق بواحدة أبسط باستخدام case:

```
#!/bin/bash

# case-menu: a menu driven system information program

clear
```

```
echo "
Please Select:

1. Display System Information
2. Display Disk Space
3. Display Home Space Utilization
0. Quit
"
read -p "Enter selection [0-3] > "

case $REPLY in
	0)	echo "Program terminated."
		exit
		;;
	1)	echo "Hostname: $HOSTNAME"
		uptime
		;;
	2)	df -h
		;;
	3)	if [[ $(id -u) -eq 0 ]]; then
			echo "Home Space Utilization (All Users)"
			du -sh /home/*
		else
			echo "Home Space Utilization ($USER)"
			du -sh $HOME
		fi
		;;
	*)	echo "Invalid entry" >&2
		exit 1
		;;
esac
```

ينظر الأمر case إلى قيمة word (في مثالنا هو REPLY) ومن ثم يحاول مطابقته بأحد "الأنماط". عند العثور على مطابقة، فستُنفَّذ الأوامر المرتبطة بذاك النمط. لن يتم الاستمرار في محاولة العثور على مطابقات بعد المطابقة لأول مرّة.

الأنماط

الأنماط التي تُستخدم مع الأمر case هي نفسها تلك التي تُستخدم مع توسعة أسماء الملفات. تنتهي الأنماط بالرمز "(". يحتوي الجدول الآتي على بعض أشكال الأنماط الصحيحة:

الجدول 31-1: أمثلة عن أنماط case

النمط	الشرح
`a)`	المطابقة إذا كانت قيمة word تساوي "a".
`[[:alpha:]])`	المطابقة إذا كانت قيمة word هي حرف أبجدي واحد.
`???)`	المطابقة إذا كان طول قيمة السلسلة النصية word هو 3.
`*.txt)`	المطابقة إذا كانت قيمة word تنتهي بالمحارف ".txt".
`*)`	مطابقة أيّة قيمة. من الجيد تضمين هذا النمط في آخر أمر case، لكي تُعالَج قيمة word التي لم تُطابَق بأحد الأنماط التي تسبقها؛ أي أنه يُستخدم لمعالجة القيم غير الصالحة.

هذا مثال عن استخدام الأنماط:

```
#!/bin/bash

read -p "Enter word > "

case $REPLY in
       [[:alpha:]])     echo  "is  a single alphabetic character." ;;
       [ABC][0-9])      echo  "is  A, B, or C followed by a digit." ;;
       ???)             echo  "is  three characters long." ;;
       *.txt)           echo  "is  a word ending in '.txt'" ;;
       *)               echo  "is  something else." ;;
esac
```

من الممكن أيضًا دمج عدّة أنماط سويةً باستخدام الخط العمودي كمحرف فصل. وهذا ما ينشئ النمط الشرطي "or". الذي هو مفيد عادةً لمعالجة الأحرف في الحالتين الكبيرة والصغيرة بآن واحد. على سبيل المثال:

```
#!/bin/bash

# case-menu: a menu driven system information program

clear
echo "
Please Select:

A. Display System Information
B. Display Disk Space
C. Display Home Space Utilization
Q. Quit
"
read -p "Enter selection [A, B, C or Q] > "

case $REPLY in
       q|Q) echo "Program terminated."
              exit
              ;;
       a|A) echo "Hostname: $HOSTNAME"
            uptime
            ;;
       b|B) df -h
            ;;
       c|C) if [[ $(id -u) -eq 0 ]]; then
                     echo "Home Space Utilization (All Users)"
                     du -sh /home/*
            else
                   echo "Home Space Utilization ($USER)"
                   du -sh $HOME
            fi
            ;;
         *) echo "Invalid entry" >&2
            exit 1
            ;;
esac
```

عدلنا برنامج case-menu لاستخدام الأحرف بدلًا من الأرقام لتحديد الاختيارات من القائمة. لاحظ كيف تسمح الأنماط باستخدام الأحرف الصغيرة أو الكبيرة.

تنفيذ أكثر من فعل

في إصدارات bash قبل 4.0، كان الأمر case يسمح بمطابقة نمط واحد فقط. وسينتهي تنفيذ الأمر case بعد مطابقة النمط. هذا مثال عن سكربت يختبر أحد المحارف لتبيين نوعه:

```
#!/bin/bash

# case4-1: test a character

read -n 1 -p "Type a character > "
echo
case $REPLY in
    [[:upper:]])   echo "'$REPLY' is upper case." ;;
    [[:lower:]])   echo "'$REPLY' is lower case." ;;
    [[:alpha:]])   echo "'$REPLY' is alphabetic." ;;
    [[:digit:]])   echo "'$REPLY' is a digit." ;;
    [[:graph:]])   echo "'$REPLY' is a visible character." ;;
    [[:punct:]])   echo "'$REPLY' is a punctuation symbol." ;;
    [[:space:]])   echo "'$REPLY' is a whitespace character." ;;
    [[:xdigit:]])  echo "'$REPLY' is a hexadecimal digit." ;;
esac
```

سيعطي تشغيل هذا السكربت الخرج الآتي:

```
[me@linuxbox ~]$ case4-1
Type a character > a
'a' is lower case.
```

يعمل السكربت عملًا صحيحًا، إلا أنه يفشل عندما يُطابِق الحرف أكثر من فئة محارف POSIX. على سبيل المثال، المحرف "a" هو حرف أبجدي، وأيضًا هو حرف صغير. لم يكن هنالك طريقة لجعل case يُطابِق أكثر من نمط في إصدارات bash أقل من 4.0. أضافت الإصدارات الحديثة من bash التعبير ";;&" الذي يُستخدم لإنهاء الأفعال، كالآتي:

```
#!/bin/bash

# case4-1: test a character

read -n 1 -p "Type a character > "
echo
case $REPLY in
    [[:upper:]])   echo "'$REPLY' is upper case." ;;&
    [[:lower:]])   echo "'$REPLY' is lower case." ;;&
    [[:alpha:]])   echo "'$REPLY' is alphabetic." ;;&
    [[:digit:]])   echo "'$REPLY' is a digit." ;;&
    [[:graph:]])   echo "'$REPLY' is a visible character." ;;&
    [[:punct:]])   echo "'$REPLY' is a punctuation symbol." ;;&
    [[:space:]])   echo "'$REPLY' is a whitespace character." ;;&
    [[:xdigit:]])  echo "'$REPLY' is a hexadecimal digit." ;;&
esac
```

سيُعطي تشغيل هذا السكربت الخرج الآتي:

```
[me@linuxbox ~]$ case4-2
Type a character > a
'a' is lower case.
'a' is alphabetic.
'a' is a visible character.
'a' is a hexadecimal digit.
```

إن إضافة "&;;" سمحت للأمر case بإكمال الاختبارات بدل التوقف عند أول مطابقة.

الخلاصة

الأمر case هو إضافة جيدة إلى معرفتنا البرمجية. حيث يستخدم لحل أنواع مُعيّنة من المشاكل كما ستلاحظ في الفصل القادم.

الفصل الثاني والثلاثون:

المعاملات الموضعية

إحدى الميزات التي كانت ناقصة في برامجنا السابقة هي إمكانية قبول وتفسير البرنامج لخيارات ووسائط سطر الأوامر. سنناقش في هذا الفصل ميزات الصدفة التي نحتاجها للوصول إلى خيارات ووسائط الأمر المُنفَّذ.

الوصول إلى مكونات الأوامر المُنفّذة

توفر الصدفة مجموعةً من المعاملات تسمى المعاملات الموضعية (Positional Parameters) تحتوى على قيم "الكلمات" الموجودة في الأمر المُنفَّذ؛ وذلك باستخدام المعاملات التي تحمل الرقم من 0 إلى 9. ويمكن شرحها بالمثال الآتي:

```
#!/bin/bash

# posit-param: script to view command line parameters

echo "
\$0 = $0
\$1 = $1
\$2 = $2
\$3 = $3
\$4 = $4
\$5 = $5
\$6 = $6
\$7 = $7
\$8 = $8
\$9 = $9
"
```

سكربت بسيط جدًا يُظهِر قيم المعاملات $0 إلى $9. عندما يُنفَّذ السكربت دون وسائط:

```
[me@linuxbox ~]$ posit-param
```

```
$0 = /home/me/bin/posit-param
$1 =
$2 =
$3 =
$4 =
$5 =
$6 =
$7 =
$8 =
$9 =
```

حتى دون تمرير وسائط إلى الأمر السابق، فستكون قيمة المعامل 0$ تساوي أول كلمة في الأمر؛ ألا وهو مسار البرنامج المُنفَّذ. أما عند تمرير وسائط للبرنامج، فسنلاحظ النتيجة الآتية:

```
[me@linuxbox ~]$ posit-param a b c d

$0 =  /home/me/bin/posit-param
$1 =  a
$2 =  b
$3 =  c
$4 =  d
$5 =
$6 =
$7 =
$8 =
$9 =
```

ملاحظة: يمكنك الوصول إلى أكثر من تسعة المعاملات وذلك بتوسعة المعاملات. إذا أردت تحديد رقم أكبر من 9، فضع ذاك الرقم بين قوسين معقوفين. على سبيل المثال: {211}$، {55}$، {10}$ وهكذا.

معرفة عدد الوسائط

توفر الصدفة المعامل #$ الذي تكون قيمته مساويةً لعدد الوسائط المُمرَّرة للبرنامج:

```
#!/bin/bash
```

```
# posit-param: script to view command line parameters

echo "
Number of arguments: $#
\$0 = $0
\$1 = $1
\$2 = $2
\$3 = $3
\$4 = $4
\$5 = $5
\$6 = $6
\$7 = $7
\$8 = $8
\$9 = $9
"
```

النتيجة:

```
[me@linuxbox ~]$ posit-param a b c d

Number of arguments: 4
$0 = /home/me/bin/posit-param
$1 = a
$2 = b
$3 = c
$4 = d
$5 =
$6 =
$7 =
$8 =
$9 =
```

الوصول إلى عدد كبير من الوسائط باستخدام shift

ما الذي سيحصل عندما نُمرِّر عددًا كبيرًا من الوسائط إلى برنامجنا السابق كما في هذا المثال:

```
[me@linuxbox ~]$ posit-param *

Number of arguments: 82
$0 = /home/me/bin/posit-param
$1 = addresses.ldif
$2 = bin
$3 = bookmarks.html
$4 = debian-500-i386-netinst.iso
$5 = debian-500-i386-netinst.jigdo
$6 = debian-500-i386-netinst.template
$7 = debian-cd_info.tar.gz
$8 = Desktop
$9 = dirlist-bin.txt
```

توسّع المحرف البديل "*" في المثال السابق إلى 82 وسيطًا. كيف نستطيع معالجة هذا الرقم الكبير من الوسائط!؟ تُوفِّر الصدفة طريقة (وإن كانت غريبة) لحل هذه المشكلة. يُحرِّك الأمر `shift` جميع الوسائط "درجة واحدة إلى الأعلى" في كل مرّة يُنفَّذ فيها. في الواقع، نستطيع أن نتعامل مع رقم وسيط واحد باستخدام `shift` (بالإضافة إلى $0 الذي لا يتغير أبدًا).

```
#!/bin/bash

# posit-param2: script to display all arguments

count=1

while [[ $# -gt 0 ]]; do
      echo "Argument $count = $1"
      count=$((count + 1))
      shift
done
```

تنتقل قيمة $2 إلى $1 ، وقيمة $3 إلى $2 وهكذا. وستنقص قيمة $# بمقدار واحد؛ وذلك عند كل تنفيذ للأمر `shift`.

أنشأنا حلقة تكرار في برنامج `posit-param2` تتحقق من عدد الوسائط المتبقية، وستستمر تلك الحلقة في التنفيذ لطالما كان هنالك وسيط واحد على الأقل. حيث نطبع قيمة الوسيط، ونزيد قيمة المتغير `count` في

كل مرة تُنفَّذ فيها حلقة التكرار للحصول على عدد الوسائط التي تمت معالجتها. ومن ثم ننفذ الأمر shift كي نجعل قيمة $1 تساوي قيمة الوسيط التالي. هذا مثال يوضح ذلك:

```
[me@linuxbox ~]$ posit-param2 a b c d
Argument 1 = a
Argument 2 = b
Argument 3 = c
Argument 4 = d
```

تطبيقات بسيطة

تستطيع كتابة تطبيقات مفيدة باستخدام الوسائط الموضعية، حتى دون استخدام shift. هذا برنامج بسيط، على سبيل المثال، يعرض معلومات عن الملفات:

```
#!/bin/bash

# file_info: simple file information program

PROGNAME=$(basename $0)

if [[ -e $1 ]]; then
      echo -e "\nFile Type:"
      file $1
      echo -e "\nFile Status:"
      stat $1
else
      echo "$PROGNAME: usage: $PROGNAME file" >&2
      exit 1
fi
```

يُظهِر البرنامج نوع الملف (وذلك باستخدام الأمر file) وحالة الملف (من أمر stat) للملف المحدد. توجد ميزة مثيرة للاهتمام في المثال السابق هي في المتغير PROGNAME الذي تُحَدَّد قيمته من ناتج الأمر basename $0. يزيل الأمر basename القسم الأول من المسار ويبقي فقط على اسم الملف. يزيل الأمر basename -في مثالنا- الجزء الأول من المسار المحتوى في المعامل $0 (المسار الكامل للبرنامج). تفيد هذه القيمة عند بناء الرسائل، كمعلومات الاستخدام التي تظهر في نهاية البرنامج... يمكن أن تعاد تسمية السكربت

وستتغير الرسالة السابقة تلقائيًا.

استخدام المعاملات الموضعية مع دوال الشِل

يمكننا استخدام المعاملات الموضعية لتمرير الوسائط إلى دوال الشِل، بنفس آلية تمريرها إلى السكربتات. سنحوِّل سكربت `file_info` إلى دالة شِل لشرح هذه الفكرة:

```
file_info () {

	# file_info: function to display file information

	if [[ -e $1 ]]; then
		echo -e "\nFile Type:"
		file $1
		echo -e "\nFile Status:"
		stat $1
	else
		echo "$FUNCNAME: usage: $FUNCNAME file" >&2
		return 1
	fi
}
```

لو استدعى السكربتُ (الذي يحتوي على دالة `file_info`) الدالةَ ومرر إليها اسم الملف كوسيط، فسيُمرَّر ذلك الاسم إلى الدالة.

نستطيع كتابة العديد من دوال الشِل المفيدة (والتي لا تُستخدم فقط في السكربتات، بل أيضًا في ملف `.bashrc`) باستخدام هذه الآلية.

لاحظ أن المتغير PROGNAME قد غُيِّر إلى متغير الصدفة FUNCNAME. ستُحدِّث الصدفة قيمة هذا المتغير تلقائيًا عند تنفيذ دوال الشِل. لاحظ أن $0 يحتوي دائمًا على المسار الكامل لأول عنصر في الأمر المُنفَّذ (اسم البرنامج) لكنه لن يحتوي على اسم دالة الشِل كما توقعنا.

التعامل مع الوسائط الموضعية كمجموعة

من المفيد في بعض الأحيان إدارة جميع الوسائط كمجموعة، على سبيل المثال، إذا أردنا كتابة "غلاف" يحيط ببرنامج آخر. هذا يعني أننا سنُنشئ سكربتًا أو دالة شِل تُبسِّط تنفيذ برنامج آخر. يوفر الغلاف قائمة بوسائط سطر الأوامر ثم يمررها إلى البرنامج.

توفر الصدفة لهذا الغرض معاملين خاصين. يتوسّع كلاهما إلى قائمة وسائط الأمر كاملةً؛ لكنهما يختلفان ببعض الجزئيّات؛ هما:

الجدول 32-1: المعاملَين الخاصَين * و @

المعامل	الشرح
$*	يتوسّع إلى قائمة بالمعاملات الموضعية، بدءًا من الرقم 1. يتوسّع، عند إضافة علامتَيّ اقتباس مزدوجتَين إلى سلسلة نصيّة محاطة بعلامتَيّ اقتباس مزدوجتَين تحتوي جميع الوسائط الموضعية؛ ويفصل بين الوسائط المحرف الأول من متغير الصدفة IFS (الذي هو، افتراضيًّا، فراغ واحد).
$@	يتوسع إلى قائمة المعاملات الموضعية بدءًا من المتغير 1. سيتوسع كل وسيط موضعي إلى كلمة محاطة بعلامتي اقتباس عندما يوضع المعامل $@ ضمن علامتي اقتباس مزدوجتين.

يُظهِر السكربت الآتي عمل هذه المعاملات:

```
#!/bin/bash

# posit-params3 : script to demonstrate $* and $@

print_params () {
      echo "\$1 = $1"
      echo "\$2 = $2"
      echo "\$3 = $3"
      echo "\$4 = $4"
}

pass_params () {
      echo -e "\n"    '$* :';    print_params   $*
      echo -e "\n"    '"$*" :';  print_params   "$*"
      echo -e "\n"    '$@ :';    print_params   $@
      echo -e "\n"    '"$@" :';  print_params   "$@"
}

pass_params "word" "words with spaces"
```

لقد أنشأنا وسطين في المثال "المعقد" السابق، هما: "word" و "words with spaces" ومررناهما إلى الدالة pass_params. ثم مررتهما تلك الدالة بدورها إلى الدالة print_params، باستخدام جميع الطرق الأربع مع المعاملين *$ و @$. سيُظهِر البرنامج الفرق بينهما عند تنفيذه:

```
[me@linuxbox ~]$ posit-param3

  $* :
$1 = word
$2 = words
$3 = with
$4 = spaces

  "$*" :
$1 = word words with spaces
$2 =
$3 =
$4 =

  $@ :
$1 = word
$2 = words
$3 = with
$4 = spaces

  "$@" :
$1 = word
$2 = words with spaces
$3 =
$4 =
```

لقد أنتج كلا المعاملين *$ و @$ أربع كلمات:

```
word words with spaces
```

أنتج "*$" "كلمة" واحدة هي:

```
"word words with spaces"
```

أما "$@" فأنتج كلمتين:

```
"word" "words with spaces"
```

الدرس الذي علينا تعلمه من المثال السابق هو أنه على الرغم من أن الصدفة توفر أربع طرق للحصول على قائمة الوسائط الموضعية، إلا أن المعامل "$@" هو أكثرها فائدةً في أغلب الحالات؛ لأنه يُبقي على كل وسيط موضعي كما هو.

برنامج أكثر كماليّةً!

سنستأنف عملنا على برنامج sys_info_page بعد غيابٍ طويل. سنضيف إليه عدّة خيارات كالآتي:

- **ملف الخرج.** سنضيف خيارًا لتحديد اسم الملف الذي سيحتوي على ناتج خرج البرنامج. يمكن تحديده بواسطة `-f file` أو `--file file`.
- **الوضع التفاعلي.** سيطلب البرنامج -عند استخدام هذا الخيار- من المستخدم إدخال اسم ملف الخرج، ثم يتحقق فيما إذا كان ذاك الملف موجودًا من قبل. إذا كان موجودًا، فسيُطلَب من المستخدم الموافقة قبل استبدال الملف. يُحدَّد هذا الخيار بواسطة `-i` أو `--interactive`.
- **المساعدة.** يمكن تحديد الخيار `-h` أو `--help` لجعل البرنامج يُظهِر معلومات الاستخدام.

هذا هو الكود اللازم لتفسير خيارات سطر الأوامر:

```
usage () {
     echo "$PROGNAME: usage: $PROGNAME [-f file | -i]"
     return
}

# process command line options

interactive=
filename=
while [[ -n $1 ]]; do
      case $1 in
            -f | --file)             shift
                                     filename=$1
                                     ;;
            -i | --interactive)      interactive=1
```

```
                                    ;;
            -h | --help)            usage
                                    exit
                                    ;;
            *)                      usage >&2
                                    exit 1
                                    ;;
      esac
      shift
done
```

لقد أضفنا دالةً تسمى usage لعرض رسالة عندما يُحَدَّد خيار المساعدة أو عند محاولة استخدام خيار غير معروف.

ثم بدأنا حلقة المعالجة. تستمر هذه الحلقة بالتنفيذ لطالما كان المعامل $1 ليس فارغًا. ويوجد الأمر shift في نهاية الحلقة كي يُحرِّك الوسائط الموضعية، ويؤدي إلى إيقاف الحلقة بعد انتهاء عملها.

لدينا داخل الحلقة عبارة case تُحدِّد فيما إذا كان الوسيط الموضعي يطابق أحد الخيارات المدعومة. إذا طابق المتغير أحد الخيارات، فسينفذ الأمر المرتبط بذاك الخيار؛ عدا ذلك، ستُطبَع رسالة خطأ وينتهي تنفيذ السكربت (مع حالة خروج تساوي الواحد).

يُعالَج الوسيط f- بطريقة مثيرة للاهتمام؛ حيث يؤدي إلى تنفيذ أمر shift إضافي عندما يُطابَق، الذي يؤدي إلى جعل قيمة $1 تساوي اسم الملف الممرر كوسيط مع الخيار f-.

سنُطبِّق الآن الكود اللازم للوضع التفاعلي:

```
# interactive mode

if [[ -n $interactive ]]; then
      while true; do
              read -p "Enter name of output file: " filename
              if [[ -e $filename ]]; then
                     read -p "'$filename' exists. Overwrite? [y/n/q] > "
                     case $REPLY in
                            Y|y) break
                                    ;;
                           Q|q)    echo "Program terminated."
```

```
                                exit
                                ;;
                        *)      continue
                                ;;
                esac
        elif [[ -z $filename ]]; then
                continue
        else
                break
        fi
    done
fi
```

إذا لم يكن المتغير interactive فارغًا، فستبدأ حلقة تكرار لا نهائية تحتوي على مِحث يطلب اسم ملف الخرج، وآلية التحقق من وجود الملف. إذا كان ملف الخرج موجود مسبقًا، فسيُخيَّر المستخدم بين الكتابة فوق الملف، أو اختيار اسم آخر، أو إنهاء البرنامج. إذا اختار المستخدم الكتابة فوق الملف، فسيُنفَّذ أمر break لإنهاء الحلقة. لاحظ كيف تختبر عبارة case فيما إذا اختار المستخدم الكتابة فوق الملف أو الخروج. لأن أي خيار آخر سيؤدي إلى استمرار الحلقة وسؤال المستخدم مرّة أُخرى.

لكي نستطيع استخدام ميّزة الإخراج إلى ملف، فعلينا أولًا تحويل الكود المسؤول عن طباعة الصفحة إلى دالة شِل، لأسبابٍ ستتضح لك بعد قليل:

```
write_html_page () {
    cat <<- _EOF_
    <HTML>
            <HEAD>
                    <TITLE>$TITLE</TITLE>
            </HEAD>
            <BODY>
                    <H1>$TITLE</H1>
                    <P>$TIMESTAMP</P>
                    $(report_uptime)
                    $(report_disk_space)
                    $(report_home_space)
            </BODY>
    </HTML>
```

```
	_EOF_
	return
}

# output html page

if [[ -n $filename ]]; then
	if touch $filename && [[ -f $filename ]]; then
		write_html_page > $filename
	else
		echo "$PROGNAME: Cannot write file '$filename'" >&2
		exit 1
	fi
else
	write_html_page
fi
```

يَظهر الكود الذي يعالج الخيار f- في آخر المثال السابق؛ حيث يختبر وجود الملف، وإن وُجِد، فسيختبر إن كان قابلًا للكتابة. سنستخدم الأمر touch للقيام بذلك؛ ثم نُتبِعه باختبار لتحديد فيما إذا كان الملف الناتج هو ملفٌ عادي. هذان الاختباران يغطيان الحالات التي يكون فيها مسار الملف غير صحيح (سيفشل تنفيذ الأمر touch)؛ وإذا كان الملف موجود مسبقًا، فهو ملف عادي.

وكما لاحظنا، تُستدعى الدالة write_html_page لتوليد الصفحة. سيُوجَّه الناتج إلى مجرى الخرج القياسي (إذا كان المتغير filename فارغًا)؛ أو سيُوجَّه إلى ملفٍ مُحدَّد.

الخلاصة

نستطيع الآن كتابة سكربتات فعّالة، وذلك بعد تعلمنا الوسائط الموضعية. تسمح الوسائط الموضوعية بكتابة دوال شِل تُوضَع في ملف bashrc. (الخاص بالمستخدم) لتنفيذ المهام البسيطة والمكررة.

لقد نمى برنامج sys_info_page وازداد تعقيدًا. هذا هو الكود الكامل له، مع تعليم التغيرات الأخيرة:

```
#!/bin/bash

# sys_info_page: program to output a system information page

```

```
PROGNAME=$(basename $0)
TITLE="System Information Report For $HOSTNAME"
CURRENT_TIME=$(date +"%x %r %Z")
TIMESTAMP="Generated $CURRENT_TIME, by $USER"
report_uptime () {
       cat <<- _EOF_
                <H2>System Uptime</H2>
                <PRE>$(uptime)</PRE>
                _EOF_
       return
}

report_disk_space () {
     cat <<- _EOF_
           <H2>Disk Space Utilization</H2>
           <PRE>$(df -h)</PRE>
           _EOF_
     return
}

report_home_space () {
     if [[ $(id -u) -eq 0 ]]; then
           cat <<- _EOF_
                 <H2>Home Space Utilization (All Users)</H2>
                 <PRE>$(du -sh /home/*)</PRE>
                 _EOF_
     else
           cat <<- _EOF_
                 <H2>Home Space Utilization ($USER)</H2>
                 <PRE>$(du -sh $HOME)</PRE>
                 _EOF_
     fi
     return
}

usage () {
```

```
     echo "$PROGNAME: usage: $PROGNAME [-f file | -i]"
     return
}
write_html_page () {
     cat <<- _EOF_
     <HTML>
           <HEAD>
                 <TITLE>$TITLE</TITLE>
           </HEAD>
           <BODY>
                 <H1>$TITLE</H1>
                 <P>$TIMESTAMP</P>
                 $(report_uptime)
                 $(report_disk_space)
                 $(report_home_space)
           </BODY>
     </HTML>
     _EOF_
     return
}

# process command line options

interactive=
filename=
while [[ -n $1 ]]; do
      case $1 in
              -f | --file)            shift
                                      filename=$1
                                      ;;
              -i | --interactive)     interactive=1
                                      ;;
              -h | --help)            usage
                                      exit
                                      ;;
              *)                      usage >&2
```

```
                                       exit 1
                                       ;;
      esac
      shift
done

# interactive mode

if [[ -n $interactive ]]; then
      while true; do
              read -p "Enter name of output file: " filename
              if [[ -e $filename ]]; then
                    read -p "'$filename' exists. Overwrite? [y/n/q] > "
                    case $REPLY in
                          Y|y) break
                                ;;
                          Q|q) echo "Program terminated."
                                exit
                                ;;
                          *)    continue
                                ;;
                    esac
              fi
      done
fi

# output html page

if [[ -n $filename ]]; then
      if touch $filename && [[ -f $filename ]]; then
              write_html_page > $filename
      else
              echo "$PROGNAME: Cannot write file '$filename'" >&2
              exit 1
      fi
else
```

```
      write_html_page
fi
```

لمّا ننتهِ بعد. توجد أشياء عديدة يمكننا القيام بها وتحسينات كثيرة نستطيع إضافتها.

الفصل الثالث والثلاثون:

بُنى التحكم: التكرار باستخدام for

سنلقي نظرة، في الفصل الأخير عن بُنى التحكم، على بينة تحكم أُخرى للتكرار. تختلف حلقة تكرار for عن حلقتَيّ تكرار while و until في أنها توفر طرقًا لمعالجة السلاسل أثناء التكرار. وهذا ما يفيد كثيرًا في البرمجة. ولهذا السبب، تُستخدَم for بكثرة عند كتابة سكربتات شِل.

تُستخدَم الحلقة for عن طريق الأمر for. يوجد شكلَين عامَين للأمر for في النسخ الحديثة من bash.

الشكل العام التقليدي لحلقة for

الشكل العام التقليدي للأمر for هو:

```
for variable [in words]; do
        commands
done
```

حيث variable هو اسم المتغير الذي سيزداد (أو يتغير) أثناء تنفيذ حلقة for، و"words "هي قائمة اختيارية بالعناصر التي ستُسنَد بشكل متسلسل إلى المتغير variable، و"commands" هي قائمة بالأوامر التي ستُنفَّذ عند كل تكرار للحلقة.

يفيد الأمر for في سطر الأوامر أيضًا. يمكننا بكل سهولة شرح طريقة عمله:

```
[me@linuxbox ~]$ for i in A B C D; do echo $i; done
A
B
C
D
```

أُعطي الأمر for، في المثال السابق، قائمةً مؤلفةً من أربع كلمات: "A"، و "B"، و "C"، و "D". ستُنفَّذ الحلقة أربع مرّات لأنها تحتوي على أربع كلمات. ستُسنَد كلمة من الكلمات الأربع إلى المتغير عند كل مرّة تنفذ فيها الحلقة.

يُظهِر الأمر echo، الموجود داخل الحلقة، قيمة المتغير i لإبراز كيفية إسناد الكلمات إليه. وكما في حلقتَيّ التكرار while و until؛ تُنهي الكلمة المحجوزة done الحلقة. الميزة الرائعة الموجودة في for تكمن

في عدد الطرق المثيرة للاهتمام التي يمكن بواسطها إنشاء قائمة الكلمات. على سبيل المثال، نستطيع القيام بذلك بتوسعة الأقواس:

```
[me@linuxbox ~]$ for i in {A..D}; do echo $i; done
A
B
C
D
```

أو توسعة المسارات:

```
[me@linuxbox ~]$ for i in distros*.txt; do echo $i; done
distros-by-date.txt
distros-dates.txt
distros-key-names.txt
distros-key-vernums.txt
distros-names.txt
distros.txt
distros-vernums.txt
distros-versions.txt
```

أو تعويض الأوامر:

```
#!/bin/bash

# longest-word : find longest string in a file

while [[ -n $1 ]]; do
       if [[ -r $1 ]]; then
              max_word=
              max_len=0
              for i in $(strings $1); do
                    len=$(echo $i | wc -c)
                    if (( len > max_len )); then
                            max_len=$len
                            max_word=$i
                    fi
```

```
                done
                echo "$1: '$max_word' ($max_len characters)"
        fi
        shift
done
```

بحثنا في المثال السابق عن أطول سلسلة نصيّة موجودة داخل ملفٍ ما. يستخدم السكربت السابق برنامج string (المُضمّن في حزمة "GNU binutils") لتوليد قائمة بجميع "الكلمات" النصية الموجودة في الملف أو الملفات (التي تُمرَّر مساراتها بواسطة سطر الأوامر). بدورها، تُعالِج الحلقة for كل كلمة في القائمة وتحدد فيما إذا كانت هي أطول كلمة. ستُطبَع أطول كلمة عند انتهاء تنفيذ حلقة التكرار.

يمكن أن يُحذَف الجزء "in words" الاختياري من أمر for. حيث يُعالِج الأمر for الوسائط الموضعية افتراضيًا. سنُعدِّل سكربت longest-word لكي يستخدم هذه الطريقة:

```
#!/bin/bash

# longest-word2 : find longest string in a file

for i; do
      if [[ -r $i ]]; then
              max_word=
              max_len=0
              for j in $(strings $i); do
                     len=$(echo $j | wc -c)
                     if (( len > max_len )); then
                            max_len=$len
                            max_word=$j
                     fi
              done
              echo "$i: '$max_word' ($max_len characters)"
      fi
done
```

كما لاحظنا، استبدلنا حلقة for بحلقة while الخارجية. وستُستخدم الوسائط الموضعية عوضًا عن قائمة الكلمات (عند حذفها). ولقد غيرنا المتغير i داخل الحلقة وجعلناه j. ولم نستخدم أيضًا الأمر shift.

لماذا i؟

ربما لاحظتَ اختيار i لاسم المتغير في حلقتَيّ for في المثالَين السابقَين. لماذا؟ لا يوجد سبب حقيقي لذلك، عدا التقاليد البرمجية. يمكنك استخدام أيّ اسم للمتغير. لكن i هو أشهرها ويتبعه j و k.

أصل تلك التقاليد هو لغة برمجة fortran. في تلك اللغة، تُعرَّف المتغيرات التي تبدأ بالأحرف I، و J، و K، و M تلقائيًا كمتغيرات تحمل أعدادًا صحيحةً. لكنَّ باقي المتغيرات التي تبدأ بباقي الأحرف، تُعرَّف كمتغيرات تحمل أعدادًا حقيقية (أي تلك الأرقام التي تحمل أجزاءً عشرية). أدّى هذا السلوك إلى جعل المبرمجين يستخدمون I، و J، و K في حلقات التكرار؛ لسهولة استخدمها عند الحاجة إلى متغيرات مؤقتة.

الشكل العام للأمر for الشبيه بلغة C

أضافت الإصدارات الأخيرة من bash، شكلًا عامًّا آخر للأمر for مُستقى من لغة برمجة C. تدعم العديد من اللغات الأُخرى هذا الشكل أيضًا:

```
for (( expression1; expression2; expression3 )); do
        commands
done
```

حيث expression1، و expression2، و expression3 هي تعابير رياضية، و commands هي الأوامر التي ستُنفَّذ عند كل تكرار للحلقة.

سلوكيًا، يشبه الشكل السابق البنية الآتية:

```
(( expression1 ))
while (( expression2 )); do
        commands
        (( expression3 ))
done
```

حيث يُستخدَم expression1 لتهيئة الشروط للحلقة، و expression2 لتحديد متى سينتهي تنفيذ الحلقة، ويُنفَّذ expression3 في نهاية كل تكرار.

هذا مثال بسيط عن استخدام شكل for الشبيه بلغة C:

```
#!/bin/bash

# simple_counter : demo of C style for command

for (( i=0; i<5; i=i+1 )); do
      echo $i
done
```

يعطي السكربت السابق النتيجة الآتية:

```
[me@linuxbox ~]$ simple_counter
0
1
2
3
4
```

في المثال السابق، هيّئ expression1 المتغير i بالقيمة 0، وسمح expression2 بإكمال الحلقة لطالما كانت قيمة i أصغر من 5، وزاد expression3 قيمة i في كل مرّة تُنفَّذ فيها الحلقة.

إن حلقة for ذات الشكل الشبيه بلغة C مفيدة عندما نحتاج إلى سلسلة عددية. سنشاهد تطبيقات كثيرة لها في الفصلَين التاليَين.

الخلاصة

بعد تعرفنا على حلقة for، يمكننا الآن إضافة بعض التحسينات إلى سكربت sys_info_page. البُنية الحالية للدالة report_home_space هي الآتية:

```
report_home_space () {
      if [[ $(id -u) -eq 0 ]]; then
              cat <<- _EOF_
                      <H2>Home Space Utilization (All Users)</H2>
                      <PRE>$(du -sh /home/*)</PRE>
                      _EOF_
      else
```

```
			cat <<- _EOF_
				<H2>Home Space Utilization ($USER)</H2>
				<PRE>$(du -sh $HOME)</PRE>
				_EOF_
	fi
	return
}
```

سنُعيد كتابتها لإظهار المزيد من المعلومات عن مجلد المنزل لجميع المستخدمين، وتضمين عدد الملفات والمجلدات الفرعية الكلي في كلٍ منها:

```
report_home_space () {
	local format="%8s%10s%10s\n"
	local i dir_list total_files total_dirs total_size user_name
	if [[ $(id -u) -eq 0 ]]; then
		dir_list=/home/*
		user_name="All Users"
	else
		dir_list=$HOME
		user_name=$USER
	fi

	echo "<H2>Home Space Utilization ($user_name)</H2>"

	for i in $dir_list; do
		total_files=$(find $i -type f | wc -l)
		total_dirs=$(find $i -type d | wc -l)
		total_size=$(du -sh $i | cut -f 1)

		echo "<H3>$i</H3>"
		echo "<PRE>"
		printf "$format" "Dirs" "Files" "Size"
		printf "$format" "----" "-----" "----"
		printf "$format" $total_dirs $total_files $total_size
		echo "</PRE>"
	done
```

```
        return
}
```

طبّقنا الكثير من ما قد تعلمناه حتى الآن في الدالة السابقة. ما زلنا نختبر إن كان المستخدم الحالي هو الجذر؛ لكن عوضًا عن القيام بمجموعة كاملة من الأفعال كجزء من `if`، فإننا سنعرّف بعض المتغيرات التي تُستخدم لاحقًا في الحلقة. لقد أضفنا عددًا من المتغيرات المحلية في الدالة واستخدمنا `printf` لتنسيق المخرجات.

الفصل الرابع والثلاثون:

السلاسل النصية والأرقام

المهمة الأساسية لبرامج الحاسوب هي التعامل مع البيانات. لقد ركزنا في الفصول السابقة على معالجة البيانات على مستوى الملفات. لكن العديد من المشاكل البرمجية تحتاج إلى واحدات أصغر من البيانات، كالسلاسل النصية والأرقام، لحلها.

سنلقي في هذا الفصل نظرةً على عدّة ميزات من ميزات الصدفة التي تُستخدم لمعالجة السلاسل النصيّة بالإضافة إلى توسعة العمليات الحسابية (التي ناقشناها بشكلٍ مبسط في الفصل السابع). هنالك أيضًا برنامجٌ شهير يسمى bc، يستطيع تنفيذ العمليات الحسابية الأكثر تعقيدًا.

توسعة المعاملات

على الرغم من أننا قد شرحنا توسعة المعاملات في الفصل السابع، إلا أننا لم نناقشها نقاشًا مفصّلًا؛ لأن أغلب عمليات توسعة المعاملات تكون في السكربتات وليس في سطر الأوامر. لقد تعاملنا من قبل مع بعض أشكال توسعة المعاملات، كمتغيرات الصدفة، على سبيل المثال.

المتغيرات البسيطة

أبسط شكل لتوسعة المعاملات يظهر جليًا عند استخدام المتغيرات، على سبيل المثال:

```
$a
```

بعد أن يُوسَّع المعامل السابق، سيُعوَّض بالقيمة التي يحملها المتغير a. يمكن أن تحاط المعاملات البسيطة بقوسَين معقوفَين:

```
${a}
```

لن يؤثر ذلك في التوسعة، لكنه ضروري إذا كان المعامل متاخمًا لنصٍ آخر، مما قد يُربِك الصدفة. سنحاول في هذا المثال إنشاء اسم ملف بإضافة محتوى السلسلة النصية "_file" إلى محتويات المتغير "$a":

```
[me@linuxbox ~]$ a="foo"
[me@linuxbox ~]$ echo "$a_file"
```

لن نحصل على أيّة نتيجة إذا جربنا المثال السابق؛ لأن الصدفة ستحاول أن تُوسِّع المتغير المسمى

"a_file" عوضًا عن "a". يمكن حل هذه المشكلة بإضافة القوسَين:

```
[me@linuxbox ~]$ echo "${a}_file"
foo_file
```

لقد لاحظنا أيضًا أن المعاملات الموضعية التي تحتل رقمًا أكبر من 9، يمكن أن نصل إليها بإحاطة الرقم بين قوسَين معقوفَين. على سبيل المثال، إذا أردنا الوصول إلى المعامل الموضعي الذي يحمل الرقم 11، فإننا نكتب:

```
${11}
```

التوسعات التي تُستخدم لمعالجة المتغيرات الفارغة

يُعالِج عدد من أشكال توسعات المعاملات المتغيرات غير الموجودة أو التي لا تحمل أيّة قيمة. تفيد هذه التوسعات عند التعامل مع معاملات موضعية غير موجودة، أو إضافة قيم افتراضية للمعاملات:

```
${parameter:-word}
```

إذا لم يكن المعامل parameter مُعرَّفًا أو كان فارغًا. فستُنتِج هذه التوسعة القيمة "word"، أما إذا حمل المعامل parameter قيمةً ما، فإن ناتج التوسعة هو تلك القيمة.

```
[me@linuxbox ~]$ foo=
[me@linuxbox ~]$ echo ${foo:-"substitute value if unset"}
substitute value if unset
[me@linuxbox ~]$ echo $foo
[me@linuxbox ~]$ foo=bar
[me@linuxbox ~]$ echo ${foo:-"substitute value if unset"}
bar
[me@linuxbox ~]$ echo $foo
bar
```

```
${parameter:=word}
```

إذا لم يكن المعامل parameter مُعرَّفًا أو كان يحمل قيمةً فارغةً، فسينتج عن هذه التوسعة القيمة "word"، بالإضافة إلى إسناد القيمة word إلى المعامل parameter. إذا لم يكن المعامل فارغًا، فسينتج عن التوسعة قيمة ذاك المعامل.

```
[me@linuxbox ~]$ foo=
```

```
[me@linuxbox ~]$ echo ${foo:="default value if unset"}
default value if unset
[me@linuxbox ~]$ echo $foo
default value if unset
[me@linuxbox ~]$ foo=bar
[me@linuxbox ~]$ echo ${foo:="default value if unset"}
bar
[me@linuxbox ~]$ echo $foo
bar
```

ملاحظة: لا يمكن إسناد قيمة المعاملات الموضعية وغيرها من المعاملات الخاصة بهذه الطريقة.

`${parameter:?word}`

إذا لم يكن المعامل parameter مُعرَّفًا أو كان فارغًا؛ فستؤدي هذه التوسعة إلى إنهاء تنفيذ السكربت مع حالة خروج لا تساوي الصفر، وستُرسَل الكلمة word إلى مجرى الخطأ القياسي. إذا لم يكن المعامل parameter فارغًا؛ فسينتج عن التوسعة قيمة ذاك المعامل.

```
[me@linuxbox ~]$ foo=
[me@linuxbox ~]$ echo ${foo:?"parameter is empty"}
bash: foo: parameter is empty
[me@linuxbox ~]$ echo $?
1
[me@linuxbox ~]$ foo=bar
[me@linuxbox ~]$ echo ${foo:?"parameter is empty"}
bar
[me@linuxbox ~]$ echo $?
0
```

`${parameter:+word}`

إذا لم كان المعامل parameter مُعرَّفًا أو كان فارغًا؛ فلن تنتج هذه التوسعة أيّ شيء. أما إذا لم يكن فارغًا، فستنتج الكلمة word، لكن القيمة المخزنة في المعامل parameter لن تتغير.

```
[me@linuxbox ~]$ foo=
[me@linuxbox ~]$ echo ${foo:+"substitute value if set"}
```

```
[me@linuxbox ~]$ foo=bar
[me@linuxbox ~]$ echo ${foo:+"substitute value if set"}
substitute value if set
```

التوسعات التي تعيد أسماء المتغيرات

تمتلك الصدفة القدرة على إعادة أسماء المتغيرات؛ لكن هذه الميزة لا تُستخدَم إلا نادرًا.

`${!prefix*}`

`${!prefix@}`

تعيد هذه التوسعة أسماء المتغيرات التي تبدأ بالجزء `prefix`. وحسب توثيق bash، فإن الشكلَين السابقَين متساويان تمامًا. هذا مثال يعرض جميع المتغيرات في البيئة التي تبدأ بالعبارة BASH:

```
[me@linuxbox ~]$ echo ${!BASH*}
BASH BASH_ARGC BASH_ARGV BASH_COMMAND BASH_COMPLETION
BASH_COMPLETION_DIR BASH_LINENO BASH_SOURCE BASH_SUBSHELL
BASH_VERSINFO BASH_VERSION
```

العمليات على السلاسل النصية

هنالك مجموعة كبيرة من التوسعات التي يمكن أن تُستخدم لإجراء عمليات معالجة على السلاسل النصية. تلائم العديد من تلك التوسعات العمليات على المسارات.

`${#parameter}`

يتوسع الشكل السابق إلى طول السلسلة النصية الموجودة في المعامل `parameter`. طبيعيًا، يكون المعامل `parameter` عبارة عن سلسلة نصية؛ لكن إذا كان "@" أو "*"، فإن ناتج عملية التوسعة هو عدد المعاملات الموضعية:

```
[me@linuxbox ~]$ foo="This string is long."
[me@linuxbox ~]$ echo "'$foo' is ${#foo} characters long."
'This string is long.' is 20 characters long.
```

`${parameter:offset}`

`${parameter:offset:length}`

تُستخدم التوسعتان السابقتان لاستخراج جزء من السلسلة النصية المحتواة في المعامل parameter. يبدأ الجزء بعد offset حرفًا من بداية السلسة النصية ويكمل حتى نهاية تلك السلسلة إلا إذا حُدِّدَ طول السلسلة المُقتطَعة (length).

```
[me@linuxbox ~]$ foo="This string is long."
[me@linuxbox ~]$ echo ${foo:5}
string is long.
[me@linuxbox ~]$ echo ${foo:5:6}
string
```

إذا كانت قيمة offset سالبةً، فهذا يعني أن السلسلة المقتطعة تبدأ من نهاية السلسلة وليس من بدايتها. لاحظ أن القيم السالبة يجب أن تُسبَق بفراغ لتجنب الخلط مع التوسعة {parameter:-word}$. لكن إذا حُدِّدَت قيمة لطول السلسلة النصية، فيجب أن تكون أكبر تمامًا من الصفر.

إذا كان المعامل هو "@"، فإن قيمة التوسعة هي عدد المعاملات الموضعية بدءًا من offset.

```
[me@linuxbox ~]$ foo="This string is long."
[me@linuxbox ~]$ echo ${foo: -5}
long.
[me@linuxbox ~]$ echo ${foo: -5:2}
lo
```

${parameter#pattern}

${parameter##pattern}

تزيل هاتان التوسعتان أول جزء من السلسلة النصية الموجودة في المعامل parameter المعرَّف بالنمط pattern. النمط pattern هو نمط محارف بديلة كتلك المستخدمة في توسعة أسماء الملفات. الاختلاف ما بين الشكلَين هو أن الشكل الذي يحتوي على "#" يحذف أصغر مطابقة، أما الشكل الذي يحتوي على "##" فيحذف أكبر مطابقة.

```
[me@linuxbox ~]$ foo=file.txt.zip
[me@linuxbox ~]$ echo ${foo#*.}
txt.zip
[me@linuxbox ~]$ echo ${foo##*.}
zip
```

`${parameter%pattern}`

`${parameter%%pattern}`

هاتان التوسعتان شبيهتان بالتوسعتين "#" و "##" في الفقرة السابقة، عدا أنهما تزيلان النص من نهاية السلسلة النصية المحتواة في المعامل parameter وليس من بدايتها.

```
[me@linuxbox ~]$ foo=file.txt.zip
[me@linuxbox ~]$ echo ${foo%.*}
file.txt
[me@linuxbox ~]$ echo ${foo%%.*}
file
```

`${parameter/pattern/string}`

`${parameter//pattern/string}`

`${parameter/#pattern/string}`

`${parameter/%pattern/string}`

تجري هذه التوسعات عملية بحث واستبدال لمحتويات المعامل parameter. إذا وُجِد نص يُطابِق نمط المحارف البديلة pattern، فسيُستبدَل ذاك النص ويُعوَّض عنه بمحتويات السلسلة النصية string. ستُستبدل أول مطابقة فقط عند استخدام الشكل العادي. وستستبدل جميع المطابقات عند استخدام الشكل "//". يتطلب الشكل "/#" أن تحدث المطابقة في بداية السلسلة النصية؛ بينما يتطلب الشكل "/%" أن تحدث المطابقة في نهاية السلسلة النصية. يمكن حذف "/string" من الشكل، مما يؤدي إلى حذف النص الذي يُطابِق النمط pattern.

```
[me@linuxbox ~]$ foo=JPG.JPG
[me@linuxbox ~]$ echo ${foo/JPG/jpg}
jpg.JPG
[me@linuxbox ~]$ echo ${foo//JPG/jpg}
jpg.jpg
[me@linuxbox ~]$ echo ${foo/#JPG/jpg}
jpg.JPG
[me@linuxbox ~]$ echo ${foo/%JPG/jpg}
JPG.jpg
```

من الجيد تعلم آلية عمل توسعات المعاملات. حيث يمكن استخدام توسعات معالجة النصوص كبدائل عن بعض الأوامر الشهيرة كالأمرَين sed و cut. يزيد استخدام التوسعات من فعالية السكربتات؛ وذلك بتقليل

عدد البرامج الخارجية المستخدمة. سنُعدِّل برنامج longest-word (الذي ناقشناه في الفصل السابق)، لكي نستخدم توسعة المعامل ${#j} عوضًا عن تعويض الأمر $(echo $j | wc -c)، كالآتي:

```
#!/bin/bash

# longest-word3 : find longest string in a file

for i; do
     if [[ -r $i ]]; then
             max_word=
             max_len=
             for j in $(strings $i); do
                     len=${#j}
                     if (( len > max_len )); then
                             max_len=$len
                             max_word=$j
                     fi
              done
              echo "$i: '$max_word' ($max_len characters)"
      fi
      shift
done
```

لنقارن فعّاليّة النسخة المُعدَّلة باستخدام الأمر time:

```
[me@linuxbox ~]$ time longest-word2 dirlist-usr-bin.txt
dirlist-usr-bin.txt: 'scrollkeeper-get-extended-content-list' (38
characters)

real 0m3.618s
user 0m1.544s
sys 0m1.768s
[me@linuxbox ~]$ time longest-word3 dirlist-usr-bin.txt
dirlist-usr-bin.txt: 'scrollkeeper-get-extended-content-list' (38
characters)

real 0m0.060s
```

```
user 0m0.056s
sys 0m0.008s
```

النسخة الأصلية من السكربت تستغرق 3.618 ثانية؛ بينما يستغرق تنفيذ النسخة الجديدة التي استخدمنا فيها التوسعة 0.06 ثانية. تحسين كبير جدًا في الفعالية!

تحويل حالة الأحرف

تدعم الإصدارات الأخيرة من bash تحويل حالة الأحرف للسلاسل النصية. لدى bash أربعة أشكال من توسعة المعاملات وخيارَين للأمر declare لدعم التحويل.

بماذا يفيد تحويل حالة الأحرف؟ اترك جانبًا القيمة الجمالية؛ هنالك دور مهم لهذه الميزة في البرمجة. لنفترض أننا نريد البحث في قاعدة بيانات. لنتخيل أن المستخدم أدخل سلسلةً نصيةً وأردنا أن نبحث عنها في قاعدة البيانات. من الممكن أن يُدخِل المستخدم القيم في حالة الأحرف الكبيرة أو الصغيرة أو أن يدمج فيما بينهما. ونحن بالطبع لا نريد أن نملأ قاعدة البيانات بكل احتمال من احتمالات حالات الأحرف؛ لذلك، ماذا بوسعنا أن نفعل؟

إحدى الطرق التي تُستخدم لحل هذه المشكلة هي "تقليل تكرار" (normalize) مدخلات المستخدم. أي أن نُحوِّل المدخلات إلى شكل معياري قبل البحث في قاعدة البيانات. نستطيع القيام بذلك بتحويل حالة جميع الأحرف إلى أحرف كبيرة أو صغيرة بما يتوافق مع المعايير المستخدمة في قاعدة البيانات.

يمكن أن يُستخدم الأمر declare لتحويل حالة أحرف السلاسل النصية إلى الحالة الكبيرة أو الصغيرة. يمكننا، باستخدام declare، أن نجعل حالة أحرف قيمة المتغير كبيرة أو صغيرة دومًا؛ دون الأخذ بعين الاعتبار حالة الأحرف الأصلية.

```
#!/bin/bash

# ul-declare: demonstrate case conversion via declare

declare -u upper
declare -l lower

if [[ $1 ]]; then
        upper="$1"
        lower="$1"
        echo $upper
```

```
            echo $lower
fi
```

استخدمنا في السكربت السابق الأمر declare لإنشاء المتغيرَين upper و lower. وأسندنا قيمة أول وسيط (المعامل الموضعي ذو الرقم 1) إلى كلا المتغيرَين ثم عرضناهما على الشاشة.

```
[me@linuxbox ~]$ ul-declare aBc
ABC
abc
```

كما لاحظنا، قد حُوِّلَت حالة أحرف الوسيط "aBc".

توجد أربع توسعات للمعاملات يمكن استخدامها لتحويل حالة الأحرف:

الجدول 34-1: التوسعات التي تُستخدم لتحويل حالة الأحرف

الصيغة	النتيجة
`${parameter,,}`	تحويل حالة قيمة المعامل parameter إلى حالة الأحرف الصغيرة.
`${parameter,}`	تحويل حالة أول حرف من قيمة المعامل parameter إلى حالة الأحرف الصغيرة.
`${parameter^^}`	تحويل حالة قيمة المعامل parameter إلى حالة الأحرف الكبيرة.
`${parameter^}`	تحويل حالة أول حرف من قيمة المعامل parameter إلى حالة الأحرف الكبيرة.

يشرح هذا سكربت تلك التوسعات:

```
#!/bin/bash

# ul-param - demonstrate case conversion via parameter expansion

if [[ $1 ]];     then
          echo   ${1,,}
          echo   ${1,}
          echo   ${1^^}
          echo   ${1^}
fi
```

وهذا مثال عن تشغيل السكربت السابق:

```
[me@linuxbox ~]$ ul-param aBc
abc
aBc
ABC
ABc
```

عالجنا أول وسيط وأظهرنا ناتج عمليات التوسعة الأربعة. وعلى الرغم من أن هذا السكربت قد استخدم المعامل الموضعي الأول، إلا أن المعامل parameter يمكن أن يكون سلسلةً نصيةً، أو متغيرًا، أو تعبيرًا نصيًّا.

العمليات الحسابية وتوسعاتها

لقد ألقينا نظرةً على توسعة العمليات الحسابية في الفصل السابع. التي تُستخدم لإجراء مختلف العمليات على الأعداد الصحيحة. شكلها العام هو:

```
$((expression))
```

حيث expression هو تعبير رياضي صحيح.

هذه التوسعة متعلقة بالأمر المركب (( )) الذي يُستخدم لاختبار صحة تعبير ما، والذي تعرفنا عليه في الفصل 27.

رأينا في الفصول السابقة بعض الأنواع الشهيرة من التعابير والمعاملات. سنناقش هنا القائمة كاملةً.

أنظمة العد

بالعودة إلى الفصل التاسع، ألقينا نظرةً على نظام العد الثماني ونظام العد الست عشري. تدعم الصدفة جميع أنظمة العد في توسعة العمليات الحسابية.

الجدول 34-2: تحديد أساس الأعداد

الشكل	الشرح
number	معاملة الأعداد التي تكون بدون أيّة إشارة خاصة على أنها أعداد في النظام العشري (ذو الأساس 10).
0number	معاملة الأعداد المسبوقة بصفر على أنها أعداد في النظام الثماني.
x0number	إشارة إلى الأعداد في النظام الست عشري.
base#number	يكون العدد مكتوبًا وفق الأساس "base".

بعض الأمثلة:

```
[me@linuxbox ~]$ echo $((0xff))
255
[me@linuxbox ~]$ echo $((2#11111111))
255
```

طبعنا، في الأمثلة السابقة، قيمة العد الست عشري ff (أكبر عدد مكون من رقمين في النظام الست عشري)، وأكبر عدد ثنائي مكون من ثماني خانات (الأساس 2).

تحديد إشارة العدد

هنالك معاملين لتحديد إشارة الأعداد، هما + و -، اللذان يشيران إلى أن العدد موجب أو سالب على الترتيب. على سبيل المثال "5-".

العمليات الحسابية البسيطة

العمليات الحسابية البسيطة مذكورة في الجدول الآتي:

الجدول 34-3: المعاملات الحسابية

المعامل	الشرح
+	جمع.
-	طرح.
*	ضرب.
/	قسمة.
**	الرفع إلى الأُس.
%	باقي القسمة.

يشرح أغلب تلك المعاملات نفسه بنفسه. لكن قسمة الأعداد الصحيحة وباقي القسمة يحتاجان إلى المناقشة.

لما كانت العمليات الحسابية التي تقوم بها الصدفة تجري على الأعداد الصحيحة؛ فإن ناتج تلك العمليات هو عدد صحيح دومًا.

```
[me@linuxbox ~]$ echo $(( 5 / 2 ))
2
```

وهذا ما يجعل حساب باقي القسمة أمرًا مهمًا:

```
[me@linuxbox ~]$ echo $(( 5 % 2 ))
1
```

بإمكاننا معرفة (باستخدام معاملي القسمة وباقي القسمة) أنّ 5 مقسومة على 2 تساوي 2 والباقي 1.

يفيد باقي القسمة في حلقات التكرار. حيث يسمح بإجراء عمليةٍ ما بعد فاصل معين. سيُطبَع، في المثال الآتي، سطرٌ يحتوي أرقامًا، وسنُعِّلم مضاعفات العدد 5:

```
#!/bin/bash

# modulo : demonstrate the modulo operator

for ((i = 0; i <= 20; i = i + 1)); do
      remainder=$((i % 5))
      if (( remainder == 0 )); then
             printf "<%d> " $i
      else
             printf "%d " $i
      fi
done
printf "\n"
```

سنحصل على النتيجة الآتية عند تنفيذ السكربت:

```
[me@linuxbox ~]$ modulo
<0> 1 2 3 4 <5> 6 7 8 9 <10> 11 12 13 14 <15> 16 17 18 19 <20>
```

الإسناد

يمكن أن تقوم التعابير الحسابية بعمليات إسناد. ربما لن تظهر فائدة استخدامها لك جليًا في الوقت الراهن. لكننا قمنا بعمليات إسناد لعدد كبير من المرات، وفي مختلف الحالات! في كل مرّة نُسند فيها قيمة ما لمتغير، فإننا نقوم بعملية إسناد. يمكننا القيام بذلك أيضًا في التعابير الرياضية:

```
[me@linuxbox ~]$ foo=
[me@linuxbox ~]$ echo $foo

[me@linuxbox ~]$ if (( foo = 5 ));then echo "It is true."; fi
It is true.
[me@linuxbox ~]$ echo $foo
5
```

أسندنا قيمةً فارغةً، في المثال السابق، إلى المتغير foo وتحققنا منها باستخدام echo. ثم استخدمنا if مع ((foo = 5)). تقوم هذه العملية بشيئَين مثيرَين للاهتمام: 1) تُسند القيمة 5 إلى المتغير foo؛ 2) تُرجِع القيمة true، لأن قيمة لا تساوي الصفر أُسندت إلى foo.

ملاحظة: من المهم أن تتذكر معنى = في التعبير السابق. إشارة = واحدة تقوم بالإسناد. 5 = foo تعني "اجعل foo مساويًا للرقم 5". بينما == تعني التحقق. 5 == foo تعني "هل قيمة foo تساوي 5؟". قد يكون الأمر مربكًا بعض الشيء؛ لأن الأمر test يقبل استخدام = للتحقق من قيمة السلاسل النصية. وهذا سبب إضافي لكي نستخدم [[ ]] و (( )) عوضًا عن test.

توفّر الصدفة رموزًا أُخرى (بالإضافة إلى =) تساعد في إنشاء عمليات إسناد مفيدة:

الجدول 34-4: معاملات الإسناد

الشكل	الشرح
parameter = value	إسناد بسيط. إسناد القيمة value إلى المعامل parameter.
parameter += value	الجمع. مكافئ للتعبير parameter = parameter + value.
parameter -= value	الطرح. مكافئ للتعبير parameter = parameter - value.
parameter *= value	الضرب. مكافئ للتعبير parameter = parameter * value.
parameter /= value	القسمة. مكافئ للتعبير parameter = parameter / value.
parameter %= value	باقي القسمة. مكافئ للتعبير parameter = parameter % value.
parameter++	إضافة الرقم 1 إلى المعامل parameter. مكافئ للشكل parameter = parameter + 1 (راجع النقاش في الأسفل).

`parameter--`	إنقاص الرقم 1 من المعامل parameter. مكافئ للشكل parameter = parameter - 1.
`++parameter`	إضافة الرقم 1 إلى المعامل parameter. مكافئ للشكل parameter = parameter + 1.
`--parameter`	إنقاص الرقم 1 من المعامل parameter. مكافئ للشكل parameter = parameter - 1.

توفر معاملات الإسناد السابقة اختصارًا للعديد من المهام الرياضية الشائعة. المعاملان ++ و -- مثيران للاهتمام. حيث يزيد المعامل ++ قيمة المتغير بمقدار واحد. بينما ينقص المعامل -- من قيمة المتغير مقدار واحد. أُخِذَ شكل هذين المعاملين من لغة برمجة C وأُستخدم من قِبل العديد من لغات البرمجة بما فيها bash.

يمكن أن يظهر المعاملان قبل أو بعد اسم المتغير. وعلى الرغم من أن كلاهما يزيد أو ينقص قيمة المتغير بمقدار واحد. إلا أن مكان وضع المعامل يؤدي إلى حدوث فرق كبير. إذا وُضِعَ المعامل قبل اسم المتغير؛ فإن المتغير سيزداد (أو ينقص) قبل إسناد القيمة إلى المتغير. أما إذا وضع بعد المتغير، فإن عملية الزيادة أو النقصان ستتم بعد إسناد القيمة إلى المتغير. قد يكون الأمر مربكًا بعض الشيء؛ لكنه سلوك مفيد وعملي. يزيل المثال الآتي الغموض عن استخدام معاملَيّ الزيادة والنقصان:

```
[me@linuxbox ~]$ foo=1
[me@linuxbox ~]$ echo $((foo++))
1
[me@linuxbox ~]$ echo $foo
2
```

إذا أسندنا القيمة 1 إلى المتغير foo ثم استخدمنا معامل الزيادة ++ بعد اسم المتغير، فإن القيمة التي ستظهر هي "1". لكن إذا ألقينا نظرةً أُخرى على قيمة المتغير، فإننا سنلاحظ أن القيمة قد ازدادت بمقدار واحد. إذا وضعنا معامل الزيادة قبل اسم المتغير، فإننا سنحصل على السلوك الذي قد توقعناه مسبقًا:

```
[me@linuxbox ~]$ foo=1
[me@linuxbox ~]$ echo $((++foo))
2
[me@linuxbox ~]$ echo $foo
2
```

سيفيد وضع المعامل قبل المتغير في أغلب الحالات.

يُستخدم عادةً معامل الزيادة (++) أو النقصان (--) مع حلقات التكرار. سنضيف بعض التحسينات إلى سكربت "باقي القسمة" السابق:

```
#!/bin/bash
# modulo2 : demonstrate the modulo operator
for ((i = 0; i <= 20; ++i )); do
      if (((i % 5) == 0 )); then
              printf "<%d> " $i
      else
              printf "%d " $i
      fi
done
printf "\n"
```

العمليات على البتات

إحدى فئات المعاملات تعالج الأرقام بطريقة غريبة. تعمل تلك المعاملات على مستوى البت. وتُستخدم لبعض المهام المنخفضة المستوى (غالبًا ما تكون قراءة أو كتابة الرايات الثنائية):

الجدول 5-34: معاملات البتات

المعامل	الشرح
~	معامل القلب. قلب جميع البتات في العدد.
<<	معامل الإزاحة نحو اليسار. إزاحة جميع البتات في العدد إلى اليسار.
<<	معامل الإزاحة نحو اليمين. إزاحة جميع البتات في العدد إلى اليمين.
&	معامل "الجمع" الثنائي. القيام بعملية "جمع" (AND) على كل البتات في عددين.
|	معامل "أو" الثنائي. القيام بعملية "أو" (OR) على جميع البتات في عددين.
^	معامل "أو الحصري". القيام بعملية "أو الحصرية" (XOR) على جميع البتات في عددين.

لاحظ وجود معاملات الإسناد لجميع المعاملات السابقة (على سبيل المثال: <<=) عدا معامل القلب.

هذا مثال يُنتِج قائمة الأعداد ذات الأس 2 باستخدام معامل الإزاحة نحو اليسار:

```
[me@linuxbox ~]$ for ((i=0;i<8;++i)); do echo $((1<<i)); done
12
48
16
32
64
128
```

العمليات المنطقية

كما اكتشفنا في الفصل 27، يدعم الأمر (( )) عددًا كبيرًا من معاملات المقارنة. لكن هناك معاملات أُخرى تُستخدم في العمليات المنطقية. يحتوي الجدول الآتي على قائمة كاملة بها:

الجدول 34-6: معاملات المقارنة

المعامل	الشرح
<=	أصغر أو يساوي.
>=	أكبر أو يساوي.
<	أصغر.
>	أكبر.
==	يساوي.
!=	لا يساوي.
&&	"و" المنطقية.
||	"أو" المنطقية.
expr1?expr2:expr3	معامل المقارنة (يسمى أيضًا معامل ternary). إذا لم تكن قيمة المتغير expr1 تساوي الصفر؛ فسيُنفَّذ expr2؛ عدا ذلك، سيُنفَّذ expr3.

تتبع التعابير قواعد المنطق الرياضي عندما تُستخدم مع المعاملات المنطقية. هذا يعني أن التعابير التي تساوي الصفر تعتبر false، والتعابير التي لا تساوي الصفر تعتبر true. يَربط الأمر (( )) النواتج بأكواد حالات الخروج العادية:

```
[me@linuxbox ~]$ if ((1)); then echo "true"; else echo "false"; fi
true
[me@linuxbox ~]$ if ((0)); then echo "true"; else echo "false"; fi
false
```

أغرب المعاملات المنطقية هو معامل المقارنة (ternary). يقوم هذا المعامل (الذي يشبه المعامل الموجود في لغة C) باختبار منطقي بشكل منفصل. يمكن أن يُستخدم كعبارة `if/then/else`. حيث يعتمد على ثلاثة تعابير رياضية (لا يسمح باستخدام السلاسل النصية). إذا كان أول متغير محقق (`true` أو لا يساوي الصفر)؛ فسيُنفَّذ التعبير الثاني؛ عدا ذلك، سيُنفَّذ التعبير الثالث. يمكننا التجربة في سطر الأوامر:

```
[me@linuxbox ~]$ a=0
[me@linuxbox ~]$ ((a<1?++a:--a))
[me@linuxbox ~]$ echo $a
1
[me@linuxbox ~]$ ((a<1?++a:--a))
[me@linuxbox ~]$ echo $a
0
```

في كل مرّة يُنفّذ فيها المعامل، في المثال السابق، ستتحول قيمة المتغير من الصفر إلى الواحد أو بالعكس.

لاحظ أنه لا يمكن القيام بعمليات إسناد ضمن التعابير مباشرةً. ستحصل على رسالة الخطأ الآتي عندما تجرّب ذلك:

```
[me@linuxbox ~]$ a=0
[me@linuxbox ~]$ ((a<1?a+=1:a-=1))
bash: ((: a<1?a+=1:a-=1: attempted assignment to non-variable (error
token is "-=1")
```

يمكن تفادي المشكلة بإحاطة تعبير الإسناد بقوسَين:

```
[me@linuxbox ~]$ ((a<1?(a+=1):(a-=1)))
```

هذا مثال كامل عن استخدام المعاملات الرياضية في سكربت ينتج جدول أعداد بسيط:

```
#!/bin/bash

# arith-loop: script to demonstrate arithmetic operators
```

```
finished=0
a=0
printf "a\ta**2\ta**3\n"
printf "=\t====\t====\n"

until ((finished)); do
	b=$((a**2))
	c=$((a**3))
	printf "%d\t%d\t%d\n" $a $b $c
	((a<10?++a:(finished=1)))
done
```

استخدمنا في السكربت السابق حلقة تكرار until تعتمد على قيمة المتغير finished. تُسنَد القيمة "0" (false) إلى المتغير عند تهيئته. وسيُكمَل تنفيذ الحلقة إلى أن تتغير قيمته إلى قيمة لا تساوي الصفر. نحسب، داخل الحلقة، مربع ومكعب المتغير a. ويتم التحقق من قيمة a في آخر الحلقة؛ إذا كانت أقل من 10، فستزداد بمقدار واحد، عدا ذلك، ستُسنَد القيمة 1 إلى المتغير (وبالتالي يصبح true) مما يؤدي إلى إنهاء الحلقة. يعطي السكربت السابق الخرج الآتي عند تنفيذه:

```
[me@linuxbox ~]$ arith-loop
a	a**2	a**3
=	====	====
0	0	0
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000
```

bc: لغة لحسابات رياضية دقيقة

لقد رأينا كيف تتعامل الصدفة مع مختلف العمليات الحسابية على الأعداد الصحيحة، لكن ماذا لو احتجنا إلى تنفيذ عمليات حسابية معقدة؟ أو أن نستخدم الأعداد ذات الفواصل العشرية؟ الجواب هو لا نستطيع ذلك! على الأقل ليس مباشرةً في الصدفة. سنحتاج إلى برنامج خارجي. توجد عدّة طرق يمكن اتباعها. إحدى تلك الطرق هي تضمين برنامج perl أو AWK، لكن لسوء الحظ، هذا الموضوع خارج عن نطاق الكتاب.

طريقة أُخرى هي استخدام برنامج حسابي متخصص. أحد تلك البرامج موجود في أغلب توزيعات لينُكس وهو bc.

يقرأ برنامج bc ملفًا مكتوبًا بشكل شبيه بلغة C ويُنفذه. يمكن أن يكون سكربت bc موجودًا في ملفٍ منفصل، أو أن يكون من مجرى الدخل القياسي. تدعم لغة bc عددًا كبيرًا من الميزات. بما فيها المتغيرات وحلقات التكرار والدوال. لكننا لن نشرح جميع ميزات bc هنا. راجع صفحة الدليل للأمر bc لمزيد من المعلومات.

لنبدأ بمثال بسيط، ليكن لدينا سكربت bc يجمع 2 مع 2:

```
/* A very simple bc script */

2 + 2
```

أول سطر من السكربت السابق هو تعليق. تستخدم لغة bc نفس نمط التعليقات المُستخدم في لغة C. يمكن أن يكون التعليق متعدد الأسطر. ويبدأ بالرمز */ وينتهي بالرمز /*.

استخدام bc

إذا حفظنا السكربت السابق في ملف foo.bc، فإننا نستطيع تنفيذه كالآتي:

```
[me@linuxbox ~]$ bc foo.bc
bc 1.06.94
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software
Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
4
```

إذا تفحصنا النص الناتج، فإننا سنرى نتيجة العملية الحسابية في الأسفل، بعد رسالة الحقوق. نستطيع إخفاء تلك الرسالة باستخدام الخيار q- (quiet).

يمكن استخدام bc تفاعليًا:

```
[me@linuxbox ~]$ bc -q
2 + 2
4
quit
```

عند استخدام bc استخدامًا تفاعليًا، فإننا سنكتب بكل بساطة العمليات الحسابية التي نود إجراءها وستظهر النتيجة مباشرةً. تُنهي التعليمة `quit` جلسة الوضع التفاعلي.

من الممكن أيضًا تمرير سكربت bc عبر مجرى الدخل القياسي.

```
[me@linuxbox ~]$ bc < foo.bc
4
```

تسمح لنا القدرة على قبول المدخلات عبر مجرى الدخل القياسي باستخدام `here documents`، و `here strings`، والأنابيب لتمرير السكربتات. هذا مثال عن استخدام `here string`:

```
[me@linuxbox ~]$ bc <<< "2+2"
4
```

سكربت تجريبي

كمثال واقعي عن الحسابات، سنُنشئ سكربتًا ينفذ عملية حسابية شائعة ألا وهي حساب دفعات القرض. سنستخدم، في السكربت الآتي، here document لتمرير السكربت إلى الأمر bc:

```
#!/bin/bash

# loan-calc : script to calculate monthly loan payments

PROGNAME=$(basename $0)

usage () {
	cat <<- EOF
	Usage: $PROGNAME PRINCIPAL INTEREST MONTHS
	Where:
```

```
	PRINCIPAL is the amount of the loan.
	INTEREST is the APR as a number (7% = 0.07).
	MONTHS is the length of the loan's term.
	EOF
}

if (($# != 3)); then
	usage
	exit 1
fi

principal=$1
interest=$2
months=$3

bc <<- EOF
	scale = 10
	i = $interest / 12
	p = $principal
	n = $months
	a = p * ((i * ((1 + i) ^ n)) / (((1 + i) ^ n) - 1))
	print a, "\n"
EOF
```

عند تنفيذه، سنحصل على النتيجة الآتية:

```
[me@linuxbox ~]$ loan-calc 135000 0.0775 180
1270.7222490000
```

يحسب المثال السابق الدفعة الشهرية لقرض بقيمة 135000 مع فائدة قدرها 7.75% لمدة 180 شهر (15 سنة). لاحظ دقة العدد الناتج. تُحَدَّد الدقة بمتغير خاص في سكربت bc هو `scale`. شرحٌ لكامل عناصر سكربت bc السابق موجودٌ في صفحة الدليل للأمر bc. وعلى الرغم من أن الشكل العام للعمليات الحسابية يختلف قليلًا عن الشكل المُتّبَع في الصدفة (يشبه bc لغة C كثيرًا)؛ لكن يجب أن يكون مألوفًا لديك، بناءً على ما تعلمته إلى الآن.

الخلاصة

لقد تعلمنا في هذا الفصل العديد من الأمور الصغيرة التي تُكوِّن حجر الأساس للسكربتات العملية. ستظهر القيمة الحقيقية لفعالية معالجة السلاسل النصية والأعداد في الصدفة عندما تزداد خبرتنا في كتابة السكربتات. يُظهِر سكربت loan-calc كيف يمكن للسكربتات البسيطة أن تقوم بأشياء مفيدة للغاية!

الفصل الخامس والثلاثون:

المصفوفات

لقد تعلمنا في الفصل السابق كيف تستطيع الصدفة معالجة السلاسل النصية والأعداد. أنواع البيانات التي تعرّفنا عليها حتى الآن تعتبر "متغيرات وحيدة القيمة"؛ أي أننا لا نستطيع إسناد أكثر من قيمة لتلك المتغيرات في آنٍ واحد.

سنلقي، في هذا الفصل، نظرةً على نوع آخر من بُنى المعطيات يسمى المصفوفة (Array)، التي تستطيع أن تحمل عدّة قيم في آنٍ واحد. توجد المصفوفات في الغالبية العظمى من لغات البرمجة؛ بما فيهم الشِل. وعلى الرغم من أن دعم المصفوفات في الصدفة ليس كاملًا كباقي لغات البرمجة؛ لكنها تبقى ذات فائدةٍ كبيرة في حلّ المشاكل البرمجية.

ما هي المصفوفات؟

المصفوفات هي متغيرات تستطيع أن تحمل أكثر من قيمة في وقتٍ واحد. تُنظّم المصفوفات كالجداول. لنأخذ مثلًا ورقة عمل (spreadsheet) كمثال عن المصفوفات. تُمثَّل ورقة عمل على أنها مصفوفة ثنائية الأبعاد. لأنها تملك صفوفًا وأعمدةً، ويمكن تحديد خلية مفردة من ورقة العمل بمعرفة سطرها وعمودها. تسلك المصفوفة نفس سلوك ورقة العمل. تمتلك المصفوفة خلايا، التي تسمى العناصر، وكل عنصر يحتوي على بيانات. يمكن الوصول إلى عنصرٍ من عناصر المصفوفة عن طريق عنوان يسمى المفتاح (index، أو subscript).

تدعم أغلب لغات البرمجة المصفوفات المتعددة الأبعاد. ورقة العمل هي مثال عن المصفوفة ذات بعدَين، العرض والارتفاع. تدعم العديد من لغات البرمجة عددًا لا نهائيًا من الأبعاد. لكن المصفوفات الثنائية والثلاثية الأبعاد هي أكثرهم استخدامًا.

إن المصفوفات في bash محدودة إلى بعد واحد فقط. يمكننا تخيلها على أنها ورقة عمل بعمود واحد. توجد العديد من التطبيقات للمصفوفات على الرغم من الدعم المحدود لها. دُعِمَت المصفوفات لأول مرّة في الإصدار الثاني من bash. لا تدعم صدفة يونكس الأصلية (sh) المصفوفات بتاتًا.

إنشاء مصفوفة

قواعد تسمية متغيرات المصفوفات كقواعد تسمية باقي المتغيرات، وسُتنشَأ تلقائيًا عند استخدمها. هذا مثال عنها:

```
[me@linuxbox ~]$ a[1]=foo
[me@linuxbox ~]$ echo ${a[1]}
foo
```

شاهدنا في المثال السابق طريقة الإسناد والوصول إلى عنصر من عناصر المصفوفة. أُسندت القيمة "foo" إلى العنصر 1 من المصفوفة a عن طريق أول أمر. أظهر الأمر الثاني القيمة الموجودة في العنصر 1. استخدام القوسين المعقوفين في الأمر الثاني ضروري لمنع الصدفة من القيام بتوسعة أسماء الملفات بدلًا من إظهار قيمة العنصر.

يمكن أن تُنشَأ المصفوفة باستخدام الأمر declare:

```
[me@linuxbox ~]$ declare -a a
```

أنشأ المثال السابق المصفوفة a عندما أُستخدِم الخيار a-.

إسناد القيم لمصفوفة

يمكن أن تُسنَد القيم بطريقتَين. يمكن إسناد القيم المفردة باستخدام الشكل الآتي:

```
name[subscript]=value
```

حيث name هو اسم المصفوفة و subscript هو عدد صحيح (أو تعبير رياضي) أكبر أو يساوي الصفر. لاحظ أن العنصر الأول من المصفوفة يحمل المفتاح 0، وليس 1. أما value، فهي القيمة التي ستُسنَد إلى عنصر المصفوفة.

يمكن إسناد عدّة قيم مباشرةً باستخدام هذا الشكل:

```
name=(value1 value2 ...)
```

حيث name هو اسم المصفوفة و value... هم القيم التي ستُسنَد بشكل متسلسل إلى عناصر المصفوفة، بدءًا من العنصر 0. إذا أردنا إسناد، على سبيل المثال، اختصارات أيام الأسبوع إلى المصفوفة days، فإننا نكتب:

```
[me@linuxbox ~]$ days=(Sun Mon Tue Wed Thu Fri Sat)
```

من الممكن أيضًا أن نُسنِد القيم إلى عنصر مُعيَّن، بتحديد المفتاح لكل قيمة:

```
[me@linuxbox ~]$ days=([0]=Sun [1]=Mon [2]=Tue [3]=Wed [4]=Thu [5]=Fri
```

```
[6]=Sat)
```

الوصول إلى عناصر المصفوفة

إذًا، ماذا تفيد المصفوفات؟ كما العديد من مهام إدارة البيانات التي يمكن القيام بها باستخدام ورقات العمل، يمكن القيام بالعديد من المهام البرمجية على المصفوفات.

سنبدأ ببرنامج بسيط لجمع وإظهار البيانات. سنُنشئ سكربتًا يتفحص أوقات التعديل لملفاتٍ موجودة في مجلدٍ مُعيّن. وسيَعرض السكربت، بالاعتماد على البيانات التي جمعها، جدولًا يُظهِر في أيّة ساعة كان آخر تعديل على الملفات. يمكن أن يُحدِّد مثل هذا السكربت أوقات نشاط النظام. يُخرِج السكربت، المسمى hours، النتيجة الآتية:

```
[me@linuxbox ~]$ hours .
Hour    Files   Hour    Files
----    -----   ----    -----
00      0       12      11
01      1       13      7
02      0       14      1
03      0       15      7
04      1       16      6
05      1       17      5
06      6       18      4
07      3       19      4
08      1       20      1
09      14      21      0
10      2       22      0
11      5       23      0
Total files = 80
```

نفّذنا البرنامج hours، محددين مجلد العمل الحالي كالمجلد الهدف. سيُولّد الجدول الظاهر أعلاه، وسيُظهَر عدد الملفات التي عُدِّلت في كل ساعة من ساعات اليوم (0-23). الكود المسؤول عن السكربت هو:

```
#!/bin/bash
# hours : script to count files by modification time
usage () {
       echo "usage: $(basename $0) directory" >&2
```

```
}
# Check that argument is a directory
if [[ ! -d $1 ]]; then
	usage
	exit 1
fi

# Initialize array
for i in {0..23}; do hours[i]=0; done
# Collect data
for i in $(stat -c %y "$1"/* | cut -c 12-13); do
	j=${i/#0}
	((++hours[j]))
	((++count))
done

# Display data
echo -e "Hour\tFiles\tHour\tFiles"
echo -e "----\t-----\t----\t-----"
for i in {0..11}; do
	j=$((i + 12))
	printf "%02d\t%d\t%02d\t%d\n" $i ${hours[i]} $j ${hours[j]}
done
printf "\nTotal files = %d\n" $count
```

يحتوي السكربت السابق على دالة واحدة (usage) وأربعة أقسام أساسية. في القسم الأول، نتحقق من وجود وسيط يُحدِّد المجلد الهدف؛ إن لم يُحَدَّد الوسيط، فستظهر رسالة الاستخدام وينتهي تنفيذ السكربت.

يُهيئ القسم الثاني المصفوفة `hours`. وذلك بإسناد القيمة 0 إلى جميع عناصر المصفوفة. ليس من الضروري تنفيذ هذه الخطوة قبل استخدام المصفوفة، لكن على السكربت التحقق من عدم وجود أي عنصر فارغ. لاحظ الطريقة المثيرة للاهتمام التي كُتِبَت فيها حلقة التكرار. تَمكّنا من إنشاء سلسلة من "الكلمات" للحلقة `for` باستخدام توسعة الأقواس (`{0..23}`).

القسم التالي يجمع المعلومات بتطبيق الأمر `stat` على كل ملف موجود في المجلد. استخدامنا `cut` للحصول على الساعة من الناتج. احتجنا، داخل الحلقة، إلى حذف الأصفار الموجودة قبل رقم الساعة، لأن السكربت سيحاول (وسيفشل) تفسير القيم من "00" إلى "09" على أنها أعداد في النظام الثماني. (راجع الجدول 34-1).

ثم زدنا قيمة عنصر المصفوفة الذي يتوافق مع ساعة التعديل. ثم في النهاية، زدنا قيمة المتغير count للحصول على العدد الإجمالي للملفات في المجلد.

يعرض آخر قسم من السكربت محتويات المصفوفة. طبعنا سطرَيّ الترويسة أولًا، ثم طبعنا قيم المصفوفة داخل حلقة تكرار. ثم أظهرنا العدد الإجمالي للملفات.

العمليات على المصفوفات

توجد العديد من العمليات الشهيرة التي يمكن أن تُطبَّق على المصفوفات. لبعض العمليات (كحذف المصفوفات، وتحديد عدد عناصرها، وترتيبها ...إلخ.) استخداماتٌ كثيرة في السكربتات العملية.

طباعة كامل محتويات مصفوفة ما

يمكن أن يُستخدم المفتاحين "*" و "@" للوصول إلى جميع العناصر في المصفوفة. وكما في المعاملات الموضعية، الرمز "@" هو أكثرهما فائدةً. هذا مثال يشرح استخدامهما:

```
[me@linuxbox ~]$ animals=("a dog" "a cat" "a fish")
[me@linuxbox ~]$ for i in ${animals[*]}; do echo $i; done
a
dog
a
cat
a
fish
[me@linuxbox ~]$ for i in ${animals[@]}; do echo $i; done
a
dog
a
cat
a
fish
[me@linuxbox ~]$ for i in "${animals[*]}"; do echo $i; done
a dog a cat a fish
[me@linuxbox ~]$ for i in "${animals[@]}"; do echo $i; done
a dog
a cat
```

```
a fish
```

أنشانا مصفوفةً باسم animals وأسندنا إليها ثلاث قيم (كل قيمة تتكون من كلمتَين). ثم نفّذنا حلقتي تكرار كـي نستكشـف تـأثير تقطيـع الكلمـات علـى عناصـر المصـفوفة. كلا الشـكلين {[*]animals}$ و {[@]animals}$ متساوي تمامًا قبل أن توضع علامتَيّ الاقتباس حولهما. حيث يؤدي الرمز "*" إلى إظهار "كلمة" واحدة تضم جميع محتويات المصفوفة. بينما أظهر الرمز "@" عناصر المصفوفة كثلاث كلمات، التي تطابق المحتوى الأصلي للمصفوفة.

تحديد عدد عناصر المصفوفة

باستخدام توسعة المعاملات، يمكننا تحديد عدد العناصر في المصفوفة بنفس الطريقة التي تُستخدَم لمعرفة طول السلاسل النصية. هذا مثال عنها:

```
[me@linuxbox ~]$ a[100]=foo
[me@linuxbox ~]$ echo ${#a[@]} # number of array elements
1
[me@linuxbox ~]$ echo ${#a[100]} # length of element 100
3
```

أنشأنا المصفوفة a وأسندنا القيمة foo إلى العنصر 100. ثم حصلنا على عدد عناصر المصفوفة باستخدام توسعة المعاملات والرمز "@". ثم ألقينا نظرة على طول العنصر 100 الذي يحتوي على السلسلة النصية "foo". من المهم ملاحظة أنه وعلى الرغم من أننا أسندنا السلسلة النصية إلى العنصر 100، لكن الصدفة أظهرت أن عدد العناصر في المصفوفة هو 1. وهذا ما يختلف عن سلوك بعض لغات البرمجة، التي تُهيّئ العناصر غير المُستخدَمة (من 0 إلى 99) بقيم فارغة.

الحصول على المفاتيح المُستخدَمة في المصفوفة

تسمح bash بوجود "فجوات" بين مفاتيح العناصر المُستخدمة، قد تكون هذه الميزة مفيدة إذا أردنا أن نحدد إذا كان أحد عناصر المصفوفة موجودًا. يمكن تحديد ذلك عن طريق توسعة المعاملات التي تستخدم الشكلَين الآتيَين:

${!*array*[*]}

${!*array*[@]}

حيث array هو اسم المتغير الحاوي على المصفوفة. وكما في باقي التوسعات التي تستخدم "*" و "@"، فإن الشكل @ المُحاط بعلامتَيّ اقتباس ذا فائدة أكبر في أكثر الحالات، لأنه سيتوسع إلى كلمات منفصلة:

```
[me@linuxbox ~]$ foo=([2]=a [4]=b [6]=c)
[me@linuxbox ~]$ for i in "${foo[@]}"; do echo $i; done
a
b
c
[me@linuxbox ~]$ for i in "${!foo[@]}"; do echo $i; done
2
4
6
```

إضافة العناصر إلى آخر المصفوفة

لا تفيدنا معرفة عدد العناصر في مصفوفة ما إذا أردنا إضافة قيم إلى آخر المصفوفة، لأن القيم المعادة (عند استخدام الرمزين * و@) لا تخبرنا بأكبر مفتاح مُستخدم في المصفوفة. لكن لحسن الحظ، تُوفِّر الصدفة حلًا لهذه المشكلة. بإمكاننا إسناد العناصر إلى آخر المصفوفة باستخدام معامل الإسناد "=+". أسندنا، في المثال الآتي، ثلاث قيم للمصفوفة foo، ثم أضفنا ثلاث قيم أُخرى:

```
[me@linuxbox ~]$ foo=(a b c)
[me@linuxbox ~]$ echo ${foo[@]}
a b c
[me@linuxbox ~]$ foo+=(d e f)
[me@linuxbox ~]$ echo ${foo[@]}
a b c d e f
```

ترتيب مصفوفة

وكما في أوراق العمل، من الضروري في أغلب الأحيان ترتيب العناصر الموجودة في عمود من البيانات. لا توجد طريقة مباشرة للقيام بذلك في الصدفة، لكن ليس من الصعب القيام بها يدويًا:

```
#!/bin/bash

# array-sort : Sort an array

a=(f e d c b a)
```

```
echo "Original array: ${a[@]}"
a_sorted=($(for i in "${a[@]}"; do echo $i; done | sort))
echo "Sorted array: ${a_sorted[@]}"
```

سيُنتج السكربت النتيجة الآتية عند تنفيذه:

```
[me@linuxbox ~]$ array-sort
Original array: f e d c b a
Sorted array:   a b c d e f
```

يعمل السكربت بنسخ محتويات المصفوفة الأصلية (a) إلى مصفوفة ثانية (a_sorted) باستخدام تعويض الأوامر. يمكن استخدام التقنية البسيطة السابقة للقيام بمختلف العمليات على المصفوفة، وذلك بتغيير الأوامر المُستخدَمة في الأنبوب.

حذف مصفوفة

يُستخدم الأمر unset لحذف مصفوفة:

```
[me@linuxbox ~]$ foo=(a b c d e f)
[me@linuxbox ~]$ echo ${foo[@]}
a b c d e f
[me@linuxbox ~]$ unset foo
[me@linuxbox ~]$ echo ${foo[@]}

[me@linuxbox ~]$
```

يمكن أن يُستخدم unset لحذف عناصر من المصفوفة:

```
[me@linuxbox ~]$ foo=(a b c d e f)
[me@linuxbox ~]$ echo ${foo[@]}
a b c d e f
[me@linuxbox ~]$ unset 'foo[2]'
[me@linuxbox ~]$ echo ${foo[@]}
a b d e f
```

حذفنا، في المثال السابق، العنصر الثالث من المصفوفة ذا المفتاح 2. تذكر أن المصفوفات تبدأ من المفتاح 0، وليس 1. لاحظ أيضًا أنه من الضروري أن يحاط عنصر المصفوفة بعلامتَيّ اقتباس، لمنع الصدفة من تنفيذ

توسعة أسماء الملفات.

وبشكل مثير للاهتمام، لا يؤدي إسناد قيمة فارغة إلى المصفوفة إلى إفراغ محتوياتها:

```
[me@linuxbox ~]$ foo=(a b c d e f)
[me@linuxbox ~]$ foo=
[me@linuxbox ~]$ echo ${foo[@]}
b c d e f
```

أيّة إشارة إلى مصفوفة ما دون استخدام مفتاح، تؤدي إلى الإشارة إلى العنصر ذي المفتاح 0 فيها:

```
[me@linuxbox ~]$ foo=(a b c d e f)
[me@linuxbox ~]$ echo ${foo[@]}
a b c d e f
[me@linuxbox ~]$ foo=A
[me@linuxbox ~]$ echo ${foo[@]}
A b c d e f
```

المصفوفات الترابطية

تدعم الإصدارة الأخيرة من bash المصفوفات الترابطية (Associative Arrays). تَستخدِم المصفوفاتُ الترابطية السلاسلَ النصية بدلًا من الأعدادِ الصحيحة كمفاتيح لعناصرها. تسمح هذه الميزة باستخدام طرق جديدة لإدارة البيانات. لننشئ، على سبيل المثال، مصفوفةً تُدعى "colors"، ولنستخدم أسماء الألوان كمفاتيح:

```
declare -A colors
colors["red"]="#ff0000"
colors["green"]="#00ff00"
colors["blue"]="#0000ff"
```

وعلى النقيض من المصفوفات ذات المفاتيح الرقمية التي تُنشأ عند استخدامها؛ يجب أن تُنشأ المصفوفة الترابطية باستخدام الأمر declare مع الخيار (الجديد) A-. يمكن الوصول إلى عناصر المصفوفات الترابطية بشكل مشابه للمصفوفات ذات المفاتيح الرقمية:

```
echo ${colors["blue"]}
```

سنلقي نظرة في الفصل القادم على سكربت يستخدم المصفوفات الترابطية لإنشاء تقرير.

الخلاصة

إذا بحثنا في صفحة bash في الدليل man عن الكلمة "array"، فسنجد العديد من الحالات التي تستخدم فيها bash المصفوفات. لكن أغلب تلك الحالات غامضة، لكنها توفر ميزات قد نحتاج إليها في حالاتٍ خاصة. في الواقع، لا تُستخدم المصفوفات كثيرًا في برمجة الشِل وسبب ذلك هو أنّ صدفات يونكس الأصلية (كصدفة sh) لا تدعم المصفوفات بتاتًا. من سوء الحظ أن المصفوفات غير مشهورة في برمجة الشِل، لأنها تُستخدم كثيرًا في لغات البرمجة الأُخرى وتُوفِّر أداة قوية لحل مختلف المشاكل البرمجية.

تُستخدم المصفوفات وحلقات التكرار مع بعضهما كثيرًا. حلقة التكرار:

```
for ((expr; expr; expr))
```

ملائمة ملائمةً كبيرةً لحساب مفاتيح المصفوفات.

الفصل السادس والثلاثون:

متفرقات

سنناقش في الفصل الأخير من رحلتنا بعض المتفرقات. وعلى الرغم من أننا، وبكل تأكيد، قد شرحنا الكثير من المواضيع في الفصول السابقة؛ لكن هنالك العديد من ميزات الصدفة bash التي لم نناقشها. أغلب تلك الميزات غامضة وتفيد الأشخاص الذين يدمجون bash في توزيعات لينُكس. لكن هنالك بعض الميزات المفيدة في بعض الحالات البرمجية (وإن لم تكن تلك الميزات شائعة الاستخدام)، سنناقشها في هذا الفصل.

إنشاء مجموعة أوامر، وصدفات فرعيّة

تسمح bash بتجميع الأوامر مع بعضها. يمكن القيام بذلك بطريقتَين: باستخدام الأمر group، أو باستخدام صدفة فرعية (subshell). هذا هو الشكل العام لكلٍ منهما:

الأمر group:

```
{ command1; command2; [command3; ...] }
```

الصدفة الفرعية:

```
(command1; command2; [command3;...])
```

يختلف الشكلان السابقان عن بعضهما بأن الأمر group يحيط مجموعة الأوامر بقوسَين معقوفَين، بينما تستخدم الصدفة الفرعية القوسين المدورَين. من المهم ملاحظة شكل استخدام الأمر group في bash، حيث يجب أن تُفصَل الأوامر عن القوسَين بفراغ، ويجب أن ينتهي آخر أمر بفاصلة منقوطة أو سطر جديد قبل قوس الإغلاق.

إذًا، بماذا يفيد تجميع الأوامر أو الصدفات الفرعية؟ على الرغم من وجود فرق مهم وجوهري بين الإثنين (سنناقشه بعد لحظات)، إلا أن كلاهما يُستخدم لإدارة إعادة التوجيه. لنفترض أنه لدينا قسم من سكربت يعيد توجيه عدّة أوامر:

```
ls -l > output.txt
echo "Listing of foo.txt" >> output.txt
cat foo.txt >> output.txt
```

مثال بسيط جدًا. يُعاد توجيه خرج ثلاثة أوامر إلى ملف مسمى output.txt. يمكن كتابة المثال السابق

كالآتي عند استخدامنا لتجميع الأوامر:

```
{ ls -l; echo "Listing of foo.txt"; cat foo.txt; } > output.txt
```

ويمكن أيضًا، وبشكلٍ مشابه، استخدام صدفة فرعيّة:

```
(ls -l; echo "Listing of foo.txt"; cat foo.txt) > output.txt
```

لقد وفرّنا بعض الكتابة عند استخدامنا لهذه التقنية. لكن الفائدة الكبرى من تجميع الأوامر والصدفات الفرعية تكون عند استخدامهما في الأنابيب. غالبًا ما يكون دمج ناتج عدّة أوامر سويةً وتمريرها إلى الأمر التالي مفيدًا. يُسهِّل تجميع الأوامر والصدفات الفرعية من ذلك:

```
{ ls -l; echo "Listing of foo.txt"; cat foo.txt; } | lpr
```

لقد مررنا ناتج ثلاثة أوامر إلى الأمر `lpr` عبر الأنبوب لإنشاء تقرير مطبوع.

سنستخدم في السكربت الآتي تجميع الأوامر وسنلقي نظرة على عدّة تقنيات برمجية يمكن أن تُستخدم مع المصفوفات الترابطية. يطبع السكربت الآتي، المسمى `array-2`، قائمةً بالملفات الموجودة في مجلد بعد تمرير مساره كوسيط، بالإضافة إلى أسماء مالكي الملفات واسم المجموعة المالكة. يطبع السكربت في نهاية القائمة إجمالي عدد الملفات التي تتعلق بكل مستخدم ومجموعة. هذه هي مخرجات السكربت عند تمرير مسار المجلد `/usr/bin` إليه (أُختصرت النتائج منعًا للإطالة):

```
[me@linuxbox ~]$ array-2 /usr/bin
/usr/bin/2to3-2.6                    root          root
/usr/bin/2to3                        root          root
/usr/bin/a2p                         root          root
/usr/bin/abrowser                    root          root
/usr/bin/aconnect                    root          root
/usr/bin/acpi_fakekey                root          root
/usr/bin/acpi_listen                 root          root
/usr/bin/add-apt-repository          root          root
.
.
.
/usr/bin/zipgrep                     root          root
/usr/bin/zipinfo                     root          root
/usr/bin/zipnote                     root          root
```

```
/usr/bin/zip                         root          root
usr/bin/zipsplit                     root          root
usr/bin/zjsdecode                    root          root
/usr/bin/zsoelim                     root          root
File owners:
daemon         :      1 file(s)
root           :   1394 file(s)

File group owners:
crontab        :      1 file(s)
daemon         :      1 file(s)
lpadmin        :      1 file(s)
mail           :      4 file(s)
mlocate        :      1 file(s)
root           :   1380 file(s)
shadow         :      2 file(s)
ssh            :      1 file(s)
tty            :      2 file(s)
utmp           :      2 file(s)
```

هذا هو السكربت المُستخدَم (مع أرقام الأسطر):

```
     1        #!/bin/bash
     2
     3        # array-2: Use arrays to tally file owners
     4
     5        declare -A files file_group file_owner groups owners
     6
     7        if [[ ! -d "$1" ]]; then
     8             echo "Usage: array-2 dir" >&2
     9             exit 1
    10        fi
    11
    12        for i in "$1"/*; do
    13             owner=$(stat -c %U "$i")
    14             group=$(stat -c %G "$i")
```

```
15            files["$i"]="$i"
16            file_owner["$i"]=$owner
17            file_group["$i"]=$group
18            ((++owners[$owner]))
19            ((++groups[$group]))
20        done
21
22        # List the collected files
23        { for i in "${files[@]}"; do
24            printf "%-40s %-10s %-10s\n" \
25                "$i" ${file_owner["$i"]} ${file_group["$i"]}
26        done } | sort
27      echo
28
29      # List owners
30      echo "File owners:"
31      { for i in "${!owners[@]}"; do
32          printf "%-10s: %5d file(s)\n" "$i" ${owners["$i"]}
33      done } | sort
34      echo
35
36      # List groups
37      echo "File group owners:"
38      { for i in "${!groups[@]}"; do
39          printf "%-10s: %5d file(s)\n" "$i" ${groups["$i"]}
40      done } | sort
```

لنلقِ نظرةً على آلية عمل السكربت:

السطر 5: يجب أن تُنشَأ المصفوفات الترابطية باستخدام الأمر `declare` مع الخيار `A-`. يُنشئ السكربت السابق خمس مصفوفات هي:

- `files`: تحتوي على أسماء الملفات الموجودة في مجلد، وتكون مفاتيحها هي أسماء الملفات.
- `file_group`: تحتوي على اسم المجموعة المالكة لكل ملف، مفاتيحها هي أسماء الملفات.
- `file_owner`: تحتوي على اسم المستخدم المالك لكل ملف، مفاتيحها هي أسماء الملفات.
- `groups`: تحتوي على عدد الملفات التي تملكها مجموعة مُعيّنة، مفاتيحها هي أسماء المجموعات.

- `owners`: تحتوي على عدد الملفات التي يملكها مُستخدم مُعيّن، مفاتيحها هي أسماء المستخدمين.

الأسطر 7-10: تتحقق من وجود المجلد الممرر كوسيط. إذا لم يكن ذاك المجلد صالحًا، فستُطبَع رسالة الاستخدام وينتهي السكربت بحالة خروج تساوي الواحد.

الأسطر 12-20: حلقة تكرار تمر على جميع الملفات الموجودة في المجلد. تستخرج الأسطر 13 و 14 اسم مالك الملف والمجموعة المالكة باستخدام الأمر `stat`، وتُسنَد القيم إلى المصفوفات الترابطية الخاصة بها (السطرين 16 و 17) باستخدام اسم الملف كمفتاح. وبشكلٍ مشابه، يُسنَد اسم الملف إلى المصفوفة `files` (السطر 15).

الأسطر 18 و 19: زيادة العدد الكلّي للملفات التي يملكها مستخدم أو مجموعة بمقدار واحد.

الأسطر 22- 27: تطبع قائمة الملفات وذلك بواسطة التوسعة `"${array[@]}"` التي تتوسع إلى قائمة بكامل عناصر المصفوفة. ويُعامَل كل عنصر ككلمة منفصلة، وهذا ما يسمح باحتواء أسماء الملفات على فراغات. لاحظ أيضًا أن كامل الحلقة محاطة بقوسَين معقوفَين لتشكيل مجموعة أوامر. وهذا ما يسمح بتمرير كامل مخرجات الحلقة إلى الأمر `sort`. وهذا ضروري، لأن عناصر المصفوفة لا تُرتَّب عند التوسعة.

الأسطر 29-40: تُشبه حلقتا التكرار الموجودتان في هذه الأسطر حلقة التكرار الأولى؛ إلا أنهما تستخدمان التوسعة `"${!array[@]}"` التي تتوسع إلى مفاتيح العناصر عوضًا عن قيمها.

استبدال العمليات

على الرغم من أن تجميع العمليات والصدفات الفرعية يشبهان بعضهما، ويُستخدمان لدمج مجاري الخرج أو الدخل أو الخطأ لإعادة توجيهها؛ إلا أن هناك فرق مهم بين تجميع العمليات والصدفات الفرعية: بينما يُنَفِّذ "تجميع الأوامر" جميع الأوامر في الصدفة الحالية، تُنَفِّذ الصدفة الفرعية (كما يوحي اسمها) جميع الأوامر في نسخة من الصدفة الحالية؛ أي أنه ستُنشَأ نسخة جديدة من الصدفة وستُنسَخ إليها البيئة. وعندما ينتهي تنفيذ الصدفة الفرعية؛ فسنفقد البيئة الخاصة بها. هذا يعني أننا سنخسر جميع التغيرات التي قمنا بها في الصدفة الفرعية بما فيها عمليات إسناد المتغيرات. وبالتالي، يفضل في أغلب الحالات استخدام تجميع الأوامر عوضًا عن الصدفات الفرعية؛ حيث تكون سرعة تنفيذ تجميع الأوامر كبيرة وتحتاج إلى ذاكرة أقل.

لقد واجهتنا مشكلة تتعلق ببيئة الصدفة الفرعية في الفصل 28، عندما اكتشفنا أن الأمر read لا يعمل عملًا صحيحًا عند استخدامه في الأنابيب. للتذكرة، إذا أنشأنا أنبوبًا كالآتي:

```
echo "foo" | read
echo $REPLY
```

فتكون قيمة المتغير REPLY فارغةً دائمةً؛ لأن الأمر read نُفِّذَ في صدفة فرعية؛ حيث ستُدمَّر نسخة المتغير

REPLY عند انتهاء تنفيذ الصدفة الفرعيّة.

ولما كانت جميع الأوامر المُستخدَمة في الأنابيب تُنَفَّذ في صدفاتٍ فرعية؛ فقد يعاني من هذه المشكلة أي أمر يقوم بإسناد قيم لمتغيرات. لحسن الحظ، تُوفِّر الصدفة توسعةً ذاتُ شكلٍ غريب تسمى "استبدال العمليات" (Process Substitution)، نستطيع استخدامها للالتفاف على هذه المشكلة.

يُعبَّر عن استبدال العمليات بطريقتين:

للعمليات (Process) التي تطبع مخرجات الأمر إلى مجرى الخرج القياسي:

```
<(list)
```

للعمليات التي تقبل المدخلات من مجرى الدخل القياسي:

```
>(list)
```

حيث `list` هي قائمة بالأوامر.

سنستخدم استبدال العمليات كالتالي كي نحلّ مشكلة read:

```
read < <(echo "foo")
echo $REPLY
```

يسمح استبدال العمليات بمعاملة خرج صدفة فرعية كملف عادي كي نستطيع إعادة توجيهه. في الواقع، لما كان استبدال العمليات هو توسعة؛ فبإمكاننا معرفة قيمته الحقيقية:

```
[me@linuxbox ~]$ echo <(echo "foo")
/dev/fd/63
```

استطعنا معرفة أن خرج الصدفة الفرعية مُخزّن في ملف مسمى `/dev/fd/63`، وذلك بواسطة الأمر echo.

يُستخدم استبدال الأوامر عادةً مع حلقات التكرار التي تحتوي على read. هذا مثال عن حلقة تكرار تستخدم read لمعالجة قائمة بمحتويات مجلد منشأة في صدفة فرعية:

```
#!/bin/bash

# pro-sub : demo of process substitution

while read attr links owner group size date time filename; do
	cat <<- EOF
		Filename:	$filename
```

```
                Size:            $size
                Owner:           $owner
                Group:           $group
                Modified:        $date $time
                Links:           $links
                Attributes:      $attr
        EOF
done < <(ls -l | tail -n +2)
```

تُنَفِّذ الحلقة الأمر read لكل سطر، الذي يحتوي على معلومات عن ملف من ملفات مجلدٍ ما. ستُنشأ القائمة في آخر سطر من السكربت. يعيد هذا السطر توجيه مخرجات توسعة استبدال العمليات إلى مجرى الدخل لحلقة التكرار. يُستخدَم الأمر tail في أنبوب استبدال العمليات كي لا يظهر أول سطر من القائمة؛ الذي لا نحتاج إليه.

سيُظهِر السكربت الخرج الآتي عند تنفيذه:

```
[me@linuxbox ~]$ pro_sub | head -n 20
Filename:       addresses.ldif
Size:           14540
Owner:          me
Group:          me
Modified:       2009-04-02 11:12
Links:          1
Attributes:     -rw-r—r--

Filename:       bin
Size:           4096
Owner:          me
Group:          me
Modified:       2009-07-10 07:31
Links:          2
Attributes:     drwxr-xr-x

Filename:       bookmarks.html
Size:           394213
Owner:          me
```

```
Group:          me
```

معالجة الإشارات

لقد شاهدنا في الفصل العاشر كيف تستجيب البرامج إلى الإشارات. يمكن إضافة هذه الميزة إلى سكربتاتنا أيضًا. وعلى الرغم من أن السكربتات التي كتبناها حتى الآن لا تحتاج إلى هذه الميزة لأنها لا تستغرق وقتًا طويلًا كي تُنَفّذ، ولأننا لا نُنشئ ملفات مؤقتة. لكن قد تستفيد السكربتات الكبيرة والمعقدة من وجود آلية لمعالجة الإشارات.

من المهم أن نأخذ بعين الاعتبار، عندما ننشئ سكربتات طويلة ومعقدة، ما الذي سيحصل إذا سجل المستخدم خروجه أو أطفئ الحاسوب في أثناء تنفيذ السكربت. عندما يحدث هكذا حدث، فستُرسَل إشارة إلى جميع العمليات. وبدورها، تقوم البرامج التي تُمثِّل تلك العمليات بأفعال لتتحقق من إنهاء البرنامج بشكل صحيح ومرتب. لنفترض، على سبيل المثال، أننا كتبنا سكربتًا ينشئ ملفًا مؤقتًا أثناء تنفيذه. يجب أن نجعل السكربت -إذا صممنا البرنامج تصميمًا جيدًا- يحذف الملف المؤقت عندما ينتهي من عمله. ومن الجيد أيضًا أن يحذف السكربت الملف إذا تلقى إشارةً تدل على ضرورة إنهاء البرنامج بشكل كامل.

توفر bash آلية لهذا الغرض تسمى `trap`. التي نستطيع استخدامها بالأمر المُضمَّن `tarp` الذي يكون شكله العام كالآتي:

```
trap argument signal [signal...]
```

حيث `argument` هي السلسلة النصية التي يجب أن تُقرأ وتعتبر أنها أمر. و `signal` هي الإشارة التي تؤدي إلى تنفيذ الأمر السابق عندما يتلقاها السكربت.

هذا مثال بسيط:

```
#!/bin/bash

# trap-demo : simple signal handling demo

trap "echo 'I am ignoring you.'" SIGINT SIGTERM

for i in {1..5}; do
	echo "Iteration $i of 5"
	sleep 5
done
```

يستخدم المثال السابق الأمر trap لتنفيذ الأمر echo عند كل مرّة يتلقى فيه السكربت إشارة SIGINT أو SIGTERM أثناء تنفيذه. هذا هو ناتج تنفيذ السكربت السابق عندما يحاول المستخدم إيقاف عمل السكربت باستخدام Ctrl-c:

```
[me@linuxbox ~]$ trap-demo
Iteration 1 of 5
Iteration 2 of 5
I am ignoring you.
Iteration 3 of 5
I am ignoring you.
Iteration 4 of 5
Iteration 5 of 5
```

كما لاحظنا، في كل مرّة يحاول فيها المستخدم إرسال إشارة إلى السكربت (بالضغط على Ctrl-c)، فستظهر رسالة عوضًا عن إنهاءه.

قد يكون من الصعب إنشاء سلسلة نصية لتشكيل مجموعة أوامر. لذا، تُستخدم عادةً دوال الشِل في هذا الصدد. في المثال الآتي، حددنا دالة منفصلة لمعالجة كل إشارة على حدة:

```
#!/bin/bash

# trap-demo2 : simple signal handling demo

exit_on_signal_SIGINT () {
       echo "Script interrupted." 2>&1
       exit 0
}

exit_on_signal_SIGTERM () {
       echo "Script terminated." 2>&1
       exit 0
}

trap exit_on_signal_SIGINT SIGINT
trap exit_on_signal_SIGTERM SIGTERM

for i in {1..5}; do
```

```
      echo "Iteration $i of 5"
      sleep 5
done
```

استخدم السكربت السابق الأمر trap مرتَين. تُنفَّذ دالة منفصلة عند تلقي إشارة معيّنة (حُدِّدَ ذلك باستخدام trap). لاحظ تضمين الأمر exit في نهاية كل دالة من الدالتَين اللتَيـن تُستخدمان لمعالجة الإشارات؛ حيث من الضروري استخدام exit لإنهاء السكربت، حيث سيكمل السكربت تنفيذه عند عدم وجودها.

سيكون ناتج السكربت السابق عندما يضغط المستخدم على Ctrl-c أثناء تنفيذه كالآتي:

```
[me@linuxbox ~]$ trap-demo2
Iteration 1 of 5
Iteration 2 of 5
Script interrupted.
```

الملفات المؤقتة

أحد أسباب تضمين معالجات الإشارات في السكربتات هو حذف الملفات المؤقتة التي يُنشئها السكربت لحفظ النتائج الوسيطة أثناء التنفيذ. هنالك شيء يشبه الفن في تسمية الملفات المؤقتة. تقليديًا، تنشئ البرامج في أنظمة يونكس ملفاتها المؤقتة في مجلد tmp/ (مجلد مشترك مخصص لهذا النوع من الملفات)؛ ولما كان هذا المجلد مشتركًا، فقد تنطوي هذه العملية على مخاطر أمنية، وخصوصًا للبرامج المُشغَّلة بامتيازات الجذر. لنترك جانبًا ضبط الأذونات المناسبة للملفات التي قد يصل إليها جميع المستخدمين في النظام؛ من المهم أن نُعطي أسماء غير متوقعة للملفات المؤقتة. وهذا ما يجنبنا الوقوع في ثغرة تسمى "temp race attack". إحدى الطرق التي تُستخدم لإنشاء اسم غير قابل للتوقع (لكنه يدل على وظيفة الملف) تشبه الطريقة الآتية:

```
tempfile=/tmp/$(basename $0).$$.$RANDOM
```

ينشئ السطر السابق اسم ملف يتضمن اسم البرنامج يتبعه رقم العملية (PID) ثم رقم عشوائي. لكن لاحظ أن متغير الصدفة RANDOM$ يعيد قيمة تتراوح بين 1-32767 فقط. وهذا ليس مجالًا كبيرًا بالنسبة إلى الحواسيب. لذلك، لا تكفي نسخة واحدة من المتغير كي نتغلب على مُهاجم مُحتمل للنظام.

طريقة أُخرى أفضل هي استخدام البرنامج mktemp (وليس دالة المكتبة المشتركة mktemp) لإنشاء الملف المؤقت. يقبل البرامج mktemp قالبًا كوسيط كي يُستخدم لبناء اسم الملف العشوائي. ويجب أن يحتوي القالب على سلسلة من حروف "X" التي ستُستبدل بأرقام وأحرف عشوائية؛ وكلما ازداد عدد

حروف "X" في القالب، كلما ازداد اسم الملف الناتج طولًا. هذا مثالٌ عن ذلك:

```
tempfile=$(mktemp /tmp/foobar.$$.XXXXXXXXXX)
```

ينشئ السطر السابق، ملفًا مؤقتًا ويسند اسمه إلى المتغير `tempfile`. ستعوض حروف "X" الموجودة في القالب بأرقام أو أحرف عشوائية. لذلك، سيكون اسم الملف النهائي (لقد ضمّننا أيضًا، في هذا المثال، قيمة المعامل الخاص $$ للحصول على رقم العملية PID) شبيهًا بالاسم الآتي:

```
/tmp/foobar.6593.UOZuvM6654
```

من الحكمة أن نتجنب استخدام المجلد `tmp/` في السكربتات التي تُنفَّذ من قِبل مستخدمين عاديين، ونستخدم عوضًا عنه مجلدًا توضع فيه الملفات المؤقتة، موجود داخل مجلد المنزل للمستخدم:

```
[[ -d $HOME/tmp ]] || mkdir $HOME/tmp
```

التنفيذ غير المتُزامن

قد نرغب في بعض الأحيان أن تُنفَّذ أكثر من مهمة واحدة في آن واحد. لقد رأينا كيف تكون جميع أنظمة التشغيل الحديثة متعددة المهام، هذا إن لم تكن متعددة المستخدمين أيضًا. يمكن بناء سكربتات كي تكون متعددة المهام.

يتم ذلك غالبًا بتشغيل سكربت أب، الذي بدوره يُشغِّل سكربت ابن (child script) واحد أو أكثر للقيام بمهمة إضافية بينما يكمل السكربت الأب تنفيذه. لكن عند تشغيل سلسلة من السكربتات بهذه الطريقة، قد تحدث مشاكل عند ربط (أو تزامن) السكربت الأب مع السكربت الابن. هذا يعني أنه: ماذا لو كان يعتمد السكربت الأب أو الابن على الآخر، ويجب أن ينتظر أحد السكربتات السكربتَ الآخر لكي ينهي عمله قبل أن يكمل السكربت الآخر تنفيذه؟

تحتوي bash أمرًا مضمنًا لإدارة مثل هذه الحالات من التنفيذ غير المتزامن. يؤدي الأمر `wait` إلى إيقاف السكربت الأب مؤقتًا حتى ينتهي تنفيذ عملية محددة (السكربت الابن).

الأمر wait

سنشرح الأمر `wait` أولًا. لكننا سنحتاج إلى سكربتَين، السكربت الأب:

```
#!/bin/bash

# async-parent : Asynchronous execution demo (parent)
```

```
echo "Parent: starting..."

echo "Parent: launching child script..."
async-child &
pid=$!
echo "Parent: child (PID= $pid) launched."

echo "Parent: continuing..."
sleep 2

echo "Parent: pausing to wait for child to finish..."
wait $pid

echo "Parent: child is finished. Continuing..."
echo "Parent: parent is done. Exiting."
```

والسكربت الابن:

```
#!/bin/bash

# async-child : Asynchronous execution demo (child)

echo "Child: child is running..."
sleep 5
echo "Child: child is done. Exiting."
```

لاحظنا، في المثال السابق، أن السكربت الابن بسيط للغاية. يبدأ "العمل الحقيقي" بواسطة السكربت الأب، يُنَفَّذ السكربت الابن في السكربت الأب، ويُنقَل إلى الخلفية. يسجِّل رقم عملية السكربت الابن بإسناد قيمة متغير الصدفة `$!` (الذي يحتوي دائمًا رقم العملية لآخر مهمة وُضِعَت في الخلفية) إلى المتغير pid.

يكمل السكربت الأب تنفيذه وينفذ الأمر `wait` مع رقم `pid` لعملية السكربت الابن. وهذا ما يؤدي إلى توقف السكربت عن العمل حتى ينتهي تنفيذ السكربت الابن، التي هي نفس النقطة التي سينتهي بعدها تنفيذ السكربت الأب.

سيُخرِج السكربت الأب والابن الناتج الآتي عندما يُنفَّذ السكربت الأب:

```
[me@linuxbox ~]$ async-parent
```

```
Parent: starting...
Parent: launching child script...
Parent: child (PID= 6741) launched.
Parent: continuing...
Child: child is running...
Parent: pausing to wait for child to finish...
Child: child is done. Exiting.
Parent: child is finished. Continuing...
Parent: parent is done. Exiting.
```

الأنابيب المسماة

يمكن إنشاء نوع خاص من الملفات في أغلب الأنظمة الشبيهة بِيونكس يدعى "الأنابيب المسماة" (Named Pipes). تُستخدم الأنابيب المسماة لإنشاء اتصال بين عمليتَين ويمكن أن تُستخدم كباقي أنواع الملفات. ليست تلك الميزة مشهورةً جدًا، لكن من الجيد تعلم آلية عملها.

هناك أسلوب برمجي شهير يسمى عميل-خادم (client-server)، يمكن أن يَستخدم طريقة اتصال كالأنابيب المُسماة، بالإضافة إلى أنواعٍ أُخرى من ما يسمى interprocess communication كالاتصالات الشبكية.

أشهر نوع من أنواع نظم الاتصال عميل-خادم هو اتصال المتصفح بخادم الوِب. يؤدي المتصفح دور العميل، ويُرسِل طلبيات إلى الخادم. ويرد الخادم على المتصفح بإرسال صفحة الوِب.

تسلك الأنابيب المسماة سلوك الملفات، لكنها تُشكِّل في الواقع حافظةً من نوع "الداخل أولًا، يخرج أولًا" (FIFO). وكما في الأنابيب العادية (غير المسماة)، تدخل البيانات من أحد الأطراف وتخرج من الطرف الآخر. من الممكن استخدام الأنابيب المسماة كالآتي:

```
process1 > named_pipe
```

و

```
process2 < named_pipe
```

وستسلك سلوك:

```
process1 | process2
```

تهيئة أنبوبة مسماة

علينا أولًا إنشاء أنبوبة مسماة. وذلك باستخدام الأمر `mkfifo`:

```
[me@linuxbox ~]$ mkfifo pipe1
[me@linuxbox ~]$ ls -l pipe1
prw-r--r-- 1 me me 0 2009-07-17 06:41 pipe1
```

استخدمنا الأمر `mkfifo` لإنشاء أنبوبة مسماة تدعى `pipe1`. شاهدنا خصائص الملف باستخدام الأمر `ls`، ولاحظنا أن أول حرف في حقل الخصائص هو "p"، الذي يشير إلى أن الملف هو أنبوبة مسماة.

استخدام الأنابيب المُسماة

سنحتاج إلى وجود نافذتَين لمحاكي الطرفية (أو طرفيتَين وهميتَين)، لكي نشرح كيف تعمل الأنابيب المسماة. سندخل الأمر البسيط الآتي في الطرفية الأولى، وسنعيد توجيه المخرجات إلى ملف الأنبوبة المُسماة:

```
[me@linuxbox ~]$ ls -l > pipe1
```

يبدو أن الأمر قد "علّق" بعد أن ضغطنا على الزر Enter. هذا بسبب عدم وجود أي شيء يستقبل البيانات في الطرف الآخر من الأنبوب. عندما تحدث تلك الحالة، نسمي الأنبوبة بأنها محجوبة (blocked). سيُزال الغموض عن الكلام السابق إذا جعلنا أحد الأوامر يقرأ من النهاية الأُخرى للأنبوب. سنُدخِل الأمر الآتي في نافذة الطرفية الثانية:

```
[me@linuxbox ~]$ cat < pipe1
```

وستظهر قائمة بمحتويات المجلد التي أُنشئت في الطرفية الأولى، في نافذة الطرفية الثانية كناتج للأمر `cat`. وسيكمل الأمر `ls` عمله ولن يبقى محجوبًا.

الخاتمة

حسنًا، لقد أكملنا رحلتنا. الشيء الوحيد الباقي عليك أن تفعله هو التدرب، ثم التدرب، ثم التدرب. وعلى الرغم من أننا شرحنا الكثير من المواضيع في رحلتنا، لكن ما فعلناه إلى الآن هو بداية مشوارنا مع سطر الأوامر! هنالك آلاف برامج سطر الأوامر التي بقي عليك استكشافها والاستمتاع بالعمل معها. ابدأ بالبحث في مجلد `/usr/bin` وسترى!

تُرِكَتْ هَذِهِ الصَفْحَةُ فَارِغَةً عَمْدًا

الملاحق

الملحق أ: مصادر إضافية

تمهيد

- بعض مقالات ويكيبيديا عن الأشخاص المشهورين الذين ذُكروا في هذا الفصل:

http://en.wikipedia.org/wiki/Linus_Torvalds

http://en.wikipedia.org/wiki/Richard_Stallman

- مؤسسة البرمجيات الحرة، ومشروع غنو:

http://en.wikipedia.org/wiki/Free_Software_Foundation

http://www.fsf.org

http://www.gnu.org

- كتب ريتشارد ستالمان مطولًا عن قضية تسمية "GNU/Linux":

http://www.gnu.org/gnu/why-gnu-linux.html

http://www.gnu.org/gnu/gnu-linux-faq.html#tools

الفصل الأول: ما هي الصَدَفة

- لمزيد من المعلومات حول Steve Bourne الذي كتب صدفة sh، راجع مقالة ويكيبيديا:

http://en.wikipedia.org/wiki/Steve_Bourne

- هذه مقالة عن مفهوم الصدفات في الحوسبة:

http://en.wikipedia.org/wiki/Shell_(computing)

الفصل الثالث: استكشاف النظام

- يمكن الحصول على النسخة الكاملة من معيار هيكلة نظام الملفات في لينُكس عبر هذا الرابط:

http://www.pathname.com/fhs/

- مقالة ويكيبيديا عن بنية المجلدات في يونكس والأنظمة الشبيهة بِيونكس:

http://en.wikipedia.org/wiki/Unix_directory_structure

- مقالة مفصّلة عن ASCII:

http://en.wikipedia.org/wiki/ASCII

الفصل الرابع: معالجة الملفات والمجلدات

- نقاش حول الوصلات الرمزية:

http://en.wikipedia.org/wiki/Symbolic_link

الفصل الخامس: التعامل مع الأوامر

هنالك العديد من المصادر التي يمكن الاستعانة بها للحصول على توثيق لينُكس أو سطر الأوامر، هذه قائمة بأفضلها:

- إن "Bash Reference Manual" هو دليل لصدفة bash. وعلى الرغم من أنه ما يزال دليلًا، إلا أنه يحتوي على أمثلة، وهو أسهل قراءةً من صفحة الدليل man:

http://www.gnu.org/software/bash/manual/bashref.html

- تحتوي صفحة الأسئلة الشائعة في bash على إجابات لكثير من التساؤلات حولها. تتوجه تلك الأسئلة إلى المستخدم المتوسط إلى المتقدم، لكنها تحتوي على الكثير من المعلومات:

http://mywiki.wooledge.org/BashFAQ

- يوفر مشروع غنو توثيقًا ضخمًا لبرامجه، التي تشكّل أساس سطر أوامر لينُكس، يمكنك الحصول على القائمة كاملةً هنا:

http://www.gnu.org/manual/manual.html

- توجد مقالةٌ جيدةٌ في ويكيبيديا عن صفحات الدليل man:

http://en.wikipedia.org/wiki/Man_page

الفصل السابع: رؤية العالم كما تراه الصدفة

- تحتوي صفحة دليل bash على أقسام تشرح التوسعات والاقتباسات بطريقة رسمية.
- يحتوي " Bash Reference Manual" أيضًا على فصول عن التوسعات والاقتباسات:

http://www.gnu.org/software/bash/manual/bashref.html

الفصل الثامن: استخدامات متقدمة للوحة المفاتيح

- توجد مقالة جيدة في ويكيبيديا عن الطرفيات:

http://en.wikipedia.org/wiki/Computer_terminal

الفصل التاسع: الأذونات

- توجد مقالة جيدة في ويكيبيديا عن البرمجيات الخبيثة:

http://en.wikipedia.org/wiki/Malware

- هنالك عددٌ من برامج سطر الأوامر التي تُستخدم لإنشاء وإدارة المستخدمين والمجموعات. للمزيد من المعلومات، راجع صفحة الدليل للأوامر الآتية:
  - `adduser`
  - `useradd`
  - `groupadd`

الفصل الحادي عشر: البيئة

- يحتوي قسم `INVOCATION` في صفحة دليل `bash` على شرحٍ لملفات البدء بجميع تفاصيلها.

الفصل الثاني عشر: مقدمة عن محرر vi

على الرغم من تعلمنا الكثير في ذاك الفصل، إلا أن ذلك هو بداية الطريق فقط! هنالك العديد من المصادر التي يمكنك الاستعانة بها كي تكمل رحلتك إلى احتراف محرر `vi`:

- كتاب الويكي "Learning The vi Editor" من ويكيبيديا، الذي يوفر دليلًا لتعلم `vi` وبعض المحررات الشبيهة به كمحرر `vim`:

http://en.wikibooks.org/wiki/Vi

- كتاب "The Vim Book" – وهو من مشروع `vim`، يشرح (تقريبًا) كل مزايا `vim`. يمكنك الحصول عليه من:

ftp://ftp.vim.org/pub/vim/doc/book/vimbook-OPL.pdf

- مقالة في ويكيبيديا عن Bill Joy، الذي أنشأ محرر `vi`:

http://en.wikipedia.org/wiki/Bill_Joy

- مقالة ويكيبيديا عن Bram Moolenaar، الذي أنشأ محرر `vim`:

http://en.wikipedia.org/wiki/Bram_Moolenaar

الفصل الثالث عشر: تخصيص المِحث

- توفر صفحة "The Bash Prompt HOWTO" من مشروع توثيق لينُكس شرحًا كاملًا لكل ما يمكن فعله مع المِحث:

http://tldp.org/HOWTO/Bash-Prompt-HOWTO/

- توجد مقالة جيدة في ويكيبيديا عن الأكواد الخاصة في ANSI:

http://en.wikipedia.org/wiki/ANSI_escape_code

الفصل الرابع عشر: إدارة الحزم

اقضِ بعض الوقت وأنت تستكشف نظام إدارة الحزم في توزيعتك. توفر كل توزيعة توثيقًا عن أدوات إدارة الحزم التي تستخدمها. يمكنك الاستزادة من هذه المصادر:

- فصل في الأسئلة الشائعة في توزيعة دبيان، يحتوي نظرةً عامةً عن إدارة الحزم في أنظمة دبيان:

http://www.debian.org/doc/FAQ/ch-pkgtools.en.html

- الصفحة الرئيسية لمشروع RPM:

http://www.rpm.org

- الصفحة الرئيسية لمشروع YUM في جامعة Duke:

http://linux.duke.edu/projects/yum/

- للحصول على بعض المعلومات العامة، راجع مقالة ويكيبيديا عن البيانات الوصفية:

http://en.wikipedia.org/wiki/Metadata

الفصل الخامس عشر: أجهزة التخزين

ألقِ نظرةً على صفحات الدليل لبعض الأوامر التي شرحناها. يدعم بعضها الكثير من الخيارات والعمليات. ابحث أيضًا عن مقالات على الإنترنت لشرح إضافة الأقراص الصلبة إلى حاسوبك، وطريقة التعامل مع الوسائط الضوئية.

الفصل السادس عشر: الشبكات

- يوفر توثيق لينُكس دليلًا لمدير شبكات لينُكس:

http://tldp.org/LDP/nag2/index.html

- تحتوي ويكيبيديا على العديد من المقالات التي تتحدث عن الشبكات:

http://en.wikipedia.org/wiki/Internet_protocol_address

http://en.wikipedia.org/wiki/Host_name

http://en.wikipedia.org/wiki/Uniform_Resource_Identifier

الفصل السابع عشر: البحث عن الملفات

- إن برامج `locate`، و `updatedb`، و `find`، و `xargs` هم جزء من حزمة `findutils` من مشروع غنو. يوفر مشروع غنو موقعًا مليئًا بالمحتوى الكثيف، ربما عليك قراءته إذا أردتَ استخدام تلك البرامج في الأنظمة عالية الحماية:

http://www.gnu.org/software/findutils/

الفصل الثامن عشر: الأرشفة والنسخ الاحتياطي

- صفحات الدليل لجميع الأوامر التي ناقشناها في ذاك الفصل تشرحها شرحًا واضحًا محتويًا على أمثلة مفيدة. بالإضافة إلى ذلك، لدى مشروع غنو دليلٌ مفيدٌ على الإنترنت يشرح `tar`. يمكن العثور عليه هنا:

http://www.gnu.org/software/tar/manual/index.html

الفصل التاسع عشر: التعابير النظامية

- هنالك العديد من المصادر الموجودة على الإنترنت لتعلم التعابير النظامية.
- تحتوي ويكيبيديا على مقالات تشرح معيارَيّ POSIX و ASCII لإعطائك بعضًا من الثقافة العامة:

http://en.wikipedia.org/wiki/Posix

http://en.wikipedia.org/wiki/Ascii

الفصل العشرون: معالجة النصوص

يحتوي موقع مشروع غنو على العديد من المقالات التي تناقش الأدوات التي شرحنها في هذا الفصل:

- من حزمة `Coreutils`:

http://www.gnu.org/software/coreutils/manual/coreutils.html#Output-of-entire-files

http://www.gnu.org/software/coreutils/manual/coreutils.html#Operating-on-sorted-files

http://www.gnu.org/software/coreutils/manual/coreutils.html#Operating-on-fields

http://www.gnu.org/software/coreutils/manual/coreutils.html#Operating-on-characters

- من حزمة `Diffutils`:

http://www.gnu.org/software/diffutils/manual/html_mono/diff.html

- `sed`:

http://www.gnu.org/software/sed/manual/sed.html

- `aspell`:

http://aspell.net/man-html/index.html

- هنالك العديد من المصادر الموجودة على الإنترنت التي تتحدث عن `sed`:

http://www.grymoire.com/Unix/Sed.html

http://sed.sourceforge.net/sed1line.txt

الفصل الحادي والعشرون: تنسيق النصوص

- دليل `groof`:

http://www.gnu.org/software/groff/manual/

- `tbl` - برنامج لتنسيق النصوص:

http://plan9.bell-labs.com/10thEdMan/tbl.pdf

- وبالتأكيد، تصفح المقالات الآتية على ويكيبيديا:

http://en.wikipedia.org/wiki/TeX

http://en.wikipedia.org/wiki/Donald_Knuth

http://en.wikipedia.org/wiki/Typesetting

الفصل الثاني والعشرون: الطباعة

- مقالة في ويكيبيديا عن لغة وصف الصفحات PostScript:

http://en.wikipedia.org/wiki/PostScript

- النظام الشائع للطباعة في يونكس:

http://en.wikipedia.org/wiki/Common_Unix_Printing_System

http://www.cups.org/

- نظم الطباعة في Berkeley و System V:

http://en.wikipedia.org/wiki/Berkeley_printing_system

http://en.wikipedia.org/wiki/System_V_printing_system

الفصل الثالث والعشرون: بناء البرامج

- هنالك مقالتان جيدتان في ويكيبيديا عن المُصرّفات والأداة make:

http://en.wikipedia.org/wiki/Compiler

http://en.wikipedia.org/wiki/Make_(software)

- دليل GNU Make:

http://www.gnu.org/software/make/manual/html_node/index.html

الفصل الرابع والعشرون: كتابة أول سكربت لك

- للحصول على برنامج "أهلًا بالعالم" في مختلف لغات البرمجة، راجع صفحة ويكيبيديا الآتية:

http://en.wikipedia.org/wiki/Hello_world

- هذه المقالة في ويكيبيديا تتحدث أكثر عن آلية عمل Shebang:

http://en.wikipedia.org/wiki/Shebang_(Unix)

الفصل الخامس والعشرون: بدء المشروع

- للمزيد من المعلومات حول HTML، راجع مقالتَيّ ويكيبيديا الآتيَين، وهذا الدليل:

http://en.wikipedia.org/wiki/Html

http://en.wikibooks.org/wiki/HTML_Programming

http://html.net/tutorials/html/

- تتضمن صفحة دليل bash قسمًا مُعَنوَنًا "HERE DOCUMENTS" يحتوي شرحًا تفصيليًا لهذه الميزة.

الفصل السادس والعشرون: نمط التصميم Top-Down

- هنالك مقالاتٌ جيدة في ويكيبيديا عن فلسفة تصميم البرمجيات، هاتان مقالتان منها:

http://en.wikipedia.org/wiki/Top-down_design

http://en.wikipedia.org/wiki/Subroutines

الفصل السابع والعشرون: بُنى التحكم: الدالة الشرطية if

- هنالك عدّة أقسام في صفحة دليل bash توفر معلومات مفصلة عن المواضيع التي شُرِحت في ذاك الفصل:
  - `Lists` (شرح عن معاملات التحكم || و &&).
  - Compound Commands (شرح عن `[[ ]]`، و `(( ))`، و `if`).
  - `CONDITIONAL EXPRESSIONS`.
  - `SHELL BUILTIN COMMANDS` (شرح عن الأمر `test`).
- تحتوي ويكيبيديا أيضًا على مقالة جيدة عن مفهوم pseudocode:

http://en.wikipedia.org/wiki/Pseudocode

الفصل الثامن والعشرون: قراءة مدخلات لوحة المفاتيح

- يحتوي "Bash Reference Manual" فصلًا عن الأوامر المضمنة في bash، التي من بينها الأمر `read`:

http://www.gnu.org/software/bash/manual/bashref.html#Bash-Builtins

الفصل التاسع والعشرون: بُنى التحكم: التكرار باستخدام while/until

- لدى "Bash Guide for Beginners" من مشروع توثيق لينُكس المزيد من الأمثلة عن حلقة تكرار while:

http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_02.html

- توجد مقالة في ويكيبيديا عن حلقات التكرار، التي هي جزءٌ من مقالةٍ أكبر عن بُنى التحكم:

http://en.wikipedia.org/wiki/Control_flow#Loops

الفصل الثلاثون: استكشاف الأخطاء وإصلاحها

- هنالك مقالتان قصيرتان في ويكيبيديا تشرحان الأخطاء البنيوية والأخطاء المنطقية:

http://en.wikipedia.org/wiki/Syntax_error

http://en.wikipedia.org/wiki/Logic_error

- هنالك العديد من المصادر على الإنترنت لشرح الجوانب التقنية للبرمجة باستخدام bash:

http://mywiki.wooledge.org/BashPitfalls

http://tldp.org/LDP/abs/html/gotchas.html

http://www.gnu.org/software/bash/manual/html_node/Reserved-Word-Index.html

- كتاب "The Art of Unix Programming" هو مرجع مهم لتعلم المفاهيم الأساسية الموجودة في برمجيات يونكس، ينطبق العديد من تلك المفاهيم على سكربتات الشِل:

http://www.faqs.org/docs/artu/

http://www.faqs.org/docs/artu/ch01s06.html

- هنالك Bash Debugger لمن يريد أن ينقح برنامجًا معقدًا:

http://bashdb.sourceforge.net/

الفصل الحادي والثلاثون: بُنى التحكم: التفرع باستخدام case

- هنالك قسمٌ في "Bash Reference Manual" عن البُنى الشرطية، يتحدث بالتفصيل عن الأمر المركب case:

http://tiswww.case.edu/php/chet/bash/bashref.html#SEC21

- يوفر دليل "Advanced Bash-Scripting Guide" المزيد من الأمثلة والتطبيقات حول case:

http://tldp.org/LDP/abs/html/testbranch.html

الفصل الثاني والثلاثون: المعاملات الموضعية

- يحتوي "Bash Hackers Wiki" مقالًا جيدًا عن المعاملات الموضعية:

http://wiki.bash-hackers.org/scripting/posparams

- يحتوي "Bash Reference Manual" مقالًا عن المعاملات الخاصة بما فيهم *$ و @$:

http://www.gnu.org/software/bash/manual/bashref.html#Special-Parameters

- بالإضافة إلى التقنيات التي شرحناها في ذاك الفصل، تحتوي bash على أمرٍ مضمنٍ آخر باسم getopts، الذي يمكن استخدامه لمعالجة وسائط سطر الأوامر. وقد شُرِح في قسم "SHELL BUILTIN COMMANDS" في صفحة دليل bash، وفي "Bash Hackers Wiki":

http://wiki.bash-hackers.org/howto/getopts_tutorial

الفصل الثالث والثلاثون: التكرار باستخدام for

- يحتوي "Advanced Bash-Scripting Guide" على فصلٍ عن التكرارات، مع العديد من الأمثلة المختلفة:

http://tldp.org/LDP/abs/html/loops1.html

- يشرح "Bash Reference Manual" الأوامر المركبة التي تُستخدم للتكرار، بما فيها for:

http://www.gnu.org/software/bash/manual/bashref.html#Looping-Constructs

الفصل الرابع والثلاثون: السلاسل النصية والأرقام

- هناك شرحٌ جيد في "Bash Hackers Wiki" عن توسعة المعاملات:

http://wiki.bash-hackers.org/syntax/pe

- ويشرحها أيضًا "Bash Reference Manual":

http://www.gnu.org/software/bash/manual/bashref.html#Shell-Parameter-Expansion

- هنالك مقالةٌ جيدةٌ في ويكيبيديا تشرح العمليات على البتات:

http://en.wikipedia.org/wiki/Bit_operation

- هنالك مقالةٌ أيضًا عن معامل المقارنة (ternary):

http://en.wikipedia.org/wiki/Ternary_operation

الفصل الخامس والثلاثون: المصفوفات

- هنالك مقالتان في ويكيبيديا تشرحان بُنى المعطيات التي ذُكِرت في ذاك الفصل:

http://en.wikipedia.org/wiki/Scalar_(computing)

http://en.wikipedia.org/wiki/Associative_array

الفصل السادس والثلاثون: متفرقات

- تحتوي صفحة دليل `bash` على قسمٍ مُعَنون "`Compound Commands`" يشرح بالتفصيل الأوامر المركبة بما فيها `group` والصدفات الفرعية.
- يحتوي القسم "`EXPANSION`" في صفحة دليل `bash` على شرحٍ عن استبدال العمليات.
- يحتوي "The Advanced Bash-Scripting Guide" أيضًا على شرحٍ عن استبدال العمليات:

http://tldp.org/LDP/abs/html/process-sub.html

- هنالك مقالتان في Linux Journal عن الأنابيب المسماة، الأولى من عام 1997:

http://www.linuxjournal.com/article/2156

- والثانية من عام 2009:

http://www.linuxjournal.com/content/using-named-pipes-fifos-bash

الملحق ب: الفهرس الهجائي